论可计算数

图灵与现代计算的诞生

深度解读人工智能领域奠基之作、图灵一生中最重要的论文

《论可计算数及其在判定问题上的应用》解开现代计算诞生背后的秘密

[美] 克里斯·伯恩哈特（Chris Bernhardt）著　雪曼 译

TURING'S VISION

THE BIRTH OF COMPUTER SCIENCE



中信出版集团 | CHINA CITIC PRESS

版权信息

书名:论可计算数：图灵与现代计算的诞生

作者:[美]克里斯·伯恩哈特

译者:雪曼

ISBN:9787508666105

中信出版集团制作发行

前言

市面上的图灵传记不胜枚举。英国演员德里克·雅各比（Derek Jacobi）将他的形象带上舞台，本尼迪克特·康伯巴奇（Benedict Cumberbatch）又在电影中进行了重新演绎。艾伦·图灵即便算不上家喻户晓的名人，也算得上众所周知的人物。很多人都知道，他在第二次世界大战期间进行的密码破译工作对盟军最终战胜德军起到了关键作用。人们可能听说过，图灵的一生以氰化物中毒悲惨而终，也有人听说过他为判断“计算机是否可以思考”而设计的测试。或许并不那么有名的事实是计算机科学界的最高奖项叫图灵奖（A. M. Turing Award）。这一奖项被奉为计算界的诺贝尔奖。每年国际计算机协会（Association for Computing Machinery，简称ACM）都会向在计算机领域有杰出贡献的人颁发图灵奖，并送上100万美元的奖金。ACM以图灵的名字命名了这一奖项，是因为图灵被视作计算机科学的奠基人之一。他做了哪些帮助人类构建计算机科学的事情？答案就是1936年图灵发表的一篇引人注目的论文，那时他只有24岁。这篇论文是图灵最重要的知识贡献。然而论文本身以及蕴藏其中的开创性观点却并没有广泛流传。本书的内容就围绕这篇论文展开。

这篇论文的题目看起来有些无趣：论可计算数及其在判定问题上的应用（*On Computable Numbers, with an Application to the Entscheidungsproblem*）。不要因为这个题目而太过沮丧，论文中包含了很多优雅而强大的结论和出色美妙的论证。图灵希望通过自己的论文证明当时一位顶尖数学家的观点是错误的。为了实现这一目标，他需要研究计算：什么是计算？我们该如何定义它？是否存在无法用计算解决的问题？他用令人眼花缭乱的技巧、别出心裁的创意回答了这些问题。

图灵仔细思考了人们完成计算的方法。他意识到，任何计算都可以拆分成一系列简单的步骤。接下来，他制造了能够完成其中每一个步骤的理论机器。这些机器就是我们现在所说的图灵机，它们能够完成任何计算[注]。而在这之后，他指出我们并不需要为每种不同算法设计各自的专属机器，只需设计一个可以运行任意算法的机器。在这一过程中，他提出了存储程序（stored-program）的概念，我们将会看到这对现代计算机的发展是至关重要的。最后，他证明了有一些特定的问题超出了计算机的能力范畴。

这些图灵机是现代计算机的理论模型。计算机能够执行的所有任务都能够由图灵机进行计算。因此，他的论文不只具有历史价值，他告诉我们计算机能完成哪些任务，不能完成哪些任务。他告诉我们计算的限制乍看起来似乎直接明了，可想要做出正确回答，却超出了任何一台计算机的能力范

畴。

这篇论文中包含的观点出现在很多大学的本科课程中，课程名称通常是计算理论（Theory of Computation）。由于大多数大学生没有选修这门课程，所以多数人并未接触过图灵的工作。总体来说，在全球庞大的人口数量中，只有很小一部分人知道图灵论文的内容。这篇论文不仅包含了很多非凡的想法，也与当代生活有着紧密联系，考虑到这些，不能不说这种陌生是一个遗憾。

广义相对论与量子力学都诞生于20世纪上半叶。对于这两大理论，大多数人脑海中都有些概念。这两大理论都建立在非常复杂的数学基础上，所以理解起来有一定的难度。不过，理解图灵的论文并没有这种压力。正如马文·明斯基（Marvin Minsky）所说："这一理论的纯粹与简明赋予其数学的美感，这种美感确保其在计算机理论中拥有永恒的地位。"[注]

我们将从图灵理论的基础开始，逐步延伸到那些惊人的结论。同时，我们也会尽可能补充图灵研究的背景和相关信息。为此，我们将介绍一些图灵论文发表前后的历史。

对于计算，不同人可能有不同的见解，这些观点并无对错之分。不同观点背后的景色迥然不同。在本书中，我们会在必要的时候稍作停留，深入探讨其中的一些观点。特别是普林斯顿大学逻辑学家阿隆佐·邱奇（Alonzo Church）的观点，它采用了一种完全不同的方式来探索计算。邱奇和图灵都曾为解答德国数学家戴维·希尔伯特（David Hilbert）提出的问题而努力研究。两人得出了一样的结论：希尔伯特做出的假设是错误的，不过邱奇先于图灵发表了自己的结论。

看到邱奇的论文时，图灵还在撰写自己的论文——知道有人已经抢先一步得出与自己一致的结论，当时图灵的心中势必满是苦涩与失望。

不过两人解决这一问题的方式却很不同，论证过程也因此大相径庭。图灵的论证要简明优雅得多。他的论文并非为了结论而发表，而是为了结论的证明过程而发表。

数学家经常会用"美丽"来形容一些出色的证明过程。在进行数学研究的时候，会有一种美学的指引。每当你做出证明却感觉过程稍显笨拙时，一定还有更好的论证有待开发。匈牙利数学家保罗·厄多斯（Paul Erdõs）在谈及《圣经》时指出，上帝在一本书中写就了所有最简捷、最美妙的论证。厄多斯说过这样一句著名的话："你不一定要信仰上帝，但是你应该信仰《圣经》。"据此来看，图灵的证明以及他参考的库尔特·哥德尔（Kurt Gödel）和格奥尔格·康托尔（Georg Cantor）的理论，一定都能被收录在

《圣经》中。

本书主要服务那些希望了解这些想法的读者。我们将从基础开始，徐徐展开整幅画卷。读者只需具备高中数学知识。本书需要认真阅读，有些章节、段落需要反复研读。因为图灵讲述的并不是计算领域一些无关紧要的细枝末节，而是一些深层次的、并不直观的内容。也就是说，很多人可能会觉得这些想法相当有趣，自己的付出也是值得的。

以下是本书中一些重要观点的概述，依据它们在本书中的出现顺序排列。

第一章

本章将关注19世纪下半叶至20世纪初的数学发展历史，介绍当时的数学基础如何摇摇欲坠，各类学者如何纷纷试图捍卫数学的基础。一位名叫戴维·希尔伯特的数学家在自己的研究项目中阐述了“判定问题”，即寻找算法来判定数学中的某些一般陈述正确与否。希尔伯特确信，这种算法一定存在。

这种观点导致了一个问题：执行一个计算，到底意味着什么？计算一直是数学的组成部分。实际上，几个世纪以来，数学和计算本质上是同义词，然而现在情况发生了转变。计算已经发展为一个独立的研究领域。图灵撰写论文的目的，就是证明希尔伯特关于计算能力的观点是错误的。

第二章

图灵希望证明存在一些超出计算机解决能力的问题。具体来说，他想要找到一个能够被证明为不可判定的判定问题。本章的目标就是解释“判定问题”和“不可判定”的含义以及它们为何重要。

我们会讨论三个著名的判定问题，这三个问题已经被证明是不可判定的。我们还会分析不可判定在这些情况中有何意义。

第三章

图灵用理论计算机器定义了计算。我们将以有限自动机为切入点研究自动机（即理论计算机器）。有限自动机比图灵机简单，并且易于描述和使用。我们会讨论有限自动机可以计算哪些问题，还会给出超越它们计算能力的例子并证明。此外，我们还会考虑如何将这些自动机执行的运算与波斯特的对应问题联系起来。

本章还会介绍正则表达式，证明它们与所说明的有限自动机的等价性。我们看到，拥有不止一种计算描述方法将十分有益。

第四章

本章介绍了图灵机。图灵机看起来并不比有限自动机复杂很多，但却被证明更加强大。上一章中的超越有限自动机计算能力的问题，被证明可以使用这些新机器解决。

本章还介绍了邱奇—图灵论题，即任意可由图灵机实现的算法。我们还将看到，这些机器带来了一种新的重要现象。一些机器永远不会停止输入，会一直运行下去。

第五章

除了图灵的方法，还有几种研究计算的不同方法。在本章，我们将讨论其中三种方法。首先是阿隆佐·邱奇的λ积分，接下来是标签系统，最后是元胞自动机。

这些计算观点看似截然不同，但每种观点都具有自身的优势。λ积分孕育了编程语言；标签系统在证明不同系统的等价性时非常有用；元胞自动机给出了计算全过程的图形表示。之后，我们将回归图灵的论证。

第六章

在之前几章中，我们一直用图形描述机器。在本章，我们将展示如何使用有限数字序列描述有限自动机和图灵机，即编码（encodings）。这与将高级语言转换为机器语言类似。编码导致了通用图灵机（Universal Turing machines）概念的产生——一个能够输入程序和数据，并在数据上运行程序的机器。我们会看到，现代计算机是通用机器。

第七章

上一章的最后一个话题讨论了在与机器描述对应的有限数字序列上运行机器，即在机器自己的编码上运行机器。这种自我引用概念相当重要。我们将详细分析并证明这个概念能够帮助我们证明某些问题是不可判定的。

本书并未假设读者具有相当强的数学知识。本章和下一章都会用到矛盾证明法。本章首先会解释这种证明方法——详细介绍“2不是有理数”的经典证明。随后，我们会介绍罗素的理发师悖论（Barber Paradox）。

介绍完这些概念后，我们将回归自动机的论证。我们证明了一些有限自动机可以接纳自己的编码，一些有限自动机则不能，但是不存在能够区分这两种情况的有限自动机。

之后，这种观点被拓展到图灵机：存在一些图灵机可以接纳自己的编码，一些图灵机则不可以，但是不存在能够区分这两种情况的图灵机。反过来，这个事实使我们可以证明某些判定问题是不可判定的。具体来说，我们证明了停机问题（Halting Problem）和接纳问题（Acceptance Problem）都是不可判定的。

第八章

图灵的论文题目为“论可计算数及其在判定问题上的应用”。我们已经介绍了判定问题，但是并未介绍可计算数。本章，我们会讨论这些数字的含义以及与它们相关的基本结论。

本章首先会介绍康托尔的基数（cardinality）概念。我们会讨论一些基本的、令人惊讶的关于无限基数的事实，还会介绍用于证明两个集合拥有不同基数的康托尔对角论证法和一般论证法。然后，我们会再次回归图灵的论文。我们还会解释有效可枚举（effectively enumerable）的含义。最后，我们会解释图灵证明可计算数并不是有效可枚举的过程。

第九章

最后一章将介绍在图灵发表论文后的几年时间里，他本人和计算机发生了什么变化。

图灵来到普林斯顿大学，在邱奇的指导下拿到了自己的博士学位。他在这里结识了约翰·冯·诺依曼（John von Neumann）。随后，图灵返回英国，并在第二次世界大战期间研究密码破译。

讲述完这些故事后，我们会简要地介绍在这40年间，现代计算机是如何从无到有的。我们将看到从复杂计算器到通用计算机，再到存储程序的通用计算的发展历程。具体来说，我们会提及图灵论文中萌生出的存储程序概念。

1950年，图灵发表了一篇论文，描述现在所谓的图灵机。我们将会简要介绍图灵机及这个概念的后续发展。

本章结尾部分将谈及杰克·科普兰（Jack Copeland）对图灵之死的研究，以及事实真相——或许图灵之死源于事故，而非谋杀。

最后，我们会谈到戈登·布朗（Gordon Brown）代表英国政府发表的道歉文章。

---

1. 图灵并未用自己的名字命名这些机器。他称它们为机器（machine），代表自动机器。阿隆佐·邱奇第一个称它们为图灵机，这一说法一直流传到现在。
2. 这段话出现在明斯基的《计算：有限和无限机器》的前言中。

第一章 背景

> 正确地审视数学，你会发现它拥有的不仅是真理，更是一种至高无上的美丽，这是一种像雕刻作品般冷峻而朴素的美。数学之美并未沾染人类的气息，也没有绘画或音乐般华丽。它极度纯净，极致完美，这种完美只有最伟大的艺术才能展现。
>
> ——伯特兰·罗素（Bertrand Russell）注

1935年，22岁的艾伦·图灵当选剑桥大学国王学院研究员。那时的他刚刚完成了数学专业的本科课程。年轻的图灵聪慧而又野心勃勃，读本科的时候就完成了中心极限定理（*Central Limit Theorem*）的证明。这个定理可能是统计学中最基础的定理，它说明了正态分布的普遍性并解释了其多样性。虽然图灵完成了证明，但他很快发现自己并不是第一个完成了这个任务的人。

早在10年之前，亚尔·瓦尔德马·林德伯格（Jarl Waldemar Lindeberg）就发表了对这一定理的证明。虽然图灵的证明并非首创，但却展现出了他过人的天分和非凡的潜力。这足以让他被剑桥选中，获得研究员的职位：这份工作为他赢得了奖金和三年食宿补助，他唯一要做的就是把精力投入数学研究之中。现在，图灵必须要证明自己。要想做到这一点，他得完成一些真正具有独创性的事情。还有什么比解决一个世界顶级数学家提出的猜想，并证明他是错误的更令人心动？这正是图灵想要做的，他将解决希尔伯特的判定问题。

在介绍图灵的工作之前，我们有必要了解希尔伯特提出这一问题的原因。这还要追溯到19世纪下半叶至20世纪上半叶数学领域的发展。我将用较多的笔墨介绍数学逻辑的兴起、人们为追求数学公理化所做的努力以及算法扮演的角色与作用。

数学的确定性

数学通常被视作“确定”的代名词。如果数学真理不是确定的，又有什么事情存在定数？纵观数学史，由于根基不牢靠而导致整个结构崩溃的案例并不少见。人类第一次感觉到数学的非确定性要追溯到公元前5世纪。据传，这种非确定性的发现也导致希帕索斯（Hippasus）因为自己证明的定理而惨遭谋杀。如今希帕索斯的证明原稿早已不知所踪，不过这段论证很

可能也会归入最美丽的数学论证之列（我们将在稍后看到完整的证明）。

古希腊数字系统由两部分组成——完整数以及完整数的比率，也就是我们现在常说的整数和有理数。希帕索斯首先假定了一个底和高都是1的直角三角形，接下来他发现这个三角形的斜边长度并不是有理数。现在看来，这完全不是问题。因为除了有理数，我们还有像π、*e*这样的无理数。我们明白希帕索斯论证中的斜边长度，实际上就是2这个无理数。然而对于古希腊人来说，这无疑是个大问题——他们的数字系统中只存在有理数。对他们来说，希帕索斯论证中那条斜边的长度无法由他们的数字表示，这也意味着还有更多的长度不是数字！

希帕索斯是毕达哥拉斯学派的成员，这个神秘的学派相信，数字能够表示所有事物的本质。由学派成员证明出数字无法表示所有长度，这无异于晴天霹雳，令人心神不安——这一论断直接撼动了他们最基础的信念。据说当希帕索斯将自己的证明展示给其他毕达哥拉斯派成员时，愤怒的同伴用沉重的链条缠住他的身体，将他溺毙在湖中央。这个故事的真实性难以考证，但无法测量的长度这一发现无疑引发了数学史上第一场地震般的剧变。

数字和长度都是基本的实体，你能够画出底和高都是1的直角三角形，意味着它的斜边是真实存在的。这条斜边拥有自己的长度，但对于古希腊人来说这却非常怪异，因为他们无法给这个长度分配一个数字。这类论证使古希腊人认为，长度才是更基础的实体。这样看来，数字确实让人有些不安——它缺少绝对的确定性。数字理论曾经被视作数学中最基本、最确定的概念，在经历了这场风云突变后，几何学很快取而代之。

从那以后直到现在，几何学都是数学教学重要的组成部分。这主要归功于一个人和一本书——欧几里得（Euclid）及其所著的《几何原本》（*Elements*）。《几何原本》是欧几里德编撰的收录了已知数学知识的百科全书，也是一本教科书。自2 000多年前写就以来，这些文字一直备受追捧，不断吸引着研究者的目光，在拜占庭和阿拉伯数学家中广为流传。1482年，《几何原本》首次印刷出版，并在之后不断再版。

欧几里德从一系列公理、假设入手[注]，逐渐延展、推理出更多新的结论。每一个新定理都能够通过公理以及之前的推论得出。

这种公理化的方法给人们带来了一种数学具有确定性的印象。如果我们知道自己使用的最初级的公理是正确的，就会知道自己的逻辑推理是有效的，这样也就能够肯定通过推理得出的结论。然而问题在于我们很容易做出一些无根据的假设——这些假设可能显而易见，甚至可能是正确的，但是却不能由最初的几条公理推导而出。当这些无根据的假设被加入证明过

程时，逻辑的有效性顷刻土崩瓦解，数学的必然性也就此丧失。

在19世纪，有人意识到欧几里德做出的很多假设并非基于自己的推导。因此，他的几何学著作需要重新修订。欧几里德在证明过程中使用的一些无法从公理中推断出的陈述，必须添加到他的公理列表之中。整个几何学的框架都需要重新扩展、更新。

戴维·希尔伯特提出了新的挑战。1899年，他发表了学术著作《几何基础》（*Grundlagen der Geometrie*），在书中列出了一个更新、更长也更完整的公理列表。希尔伯特十分审慎，希望确保没有根据的假设不会混入证明过程。为了实现这一目标，他对公理的定义与欧几里德的定义有着非常大的差异。

在欧几里德看来，像“两点确定一条直线”这样的公理是不证自明的。“点”和“线”都有自身的含义。希尔伯特的方法却不同，他意识到任何公理和定义系统都应该始于某个起点，这些最初的陈述中势必会包含此前从未被定义过的术语。

对希尔伯特来说，公理是能够用来证明其他观点的陈述，但公理并不能被视作不证自明的真理。欧几里德的公理“两点确定一条直线”中包含了“点”和“线”这两个没有被定义过的概念，因此这两个字不应该具有任何意义。公理会定义这些未定义概念之间的关系。正如希尔伯特指出的，由于这些术语并没有任何意义，因此理论上你可以选用任何一个词来替换“点”和“线”。据说，他曾将公理中的“点”、“线”、“面”等字眼，换作“桌子”、“椅子”、“几杯啤酒”。

伯特兰·罗素曾经颇为风趣地对此做出总结：“数学可能就是这样一个学科，我们可能永远不知道自己在谈论什么，或者无法判断自己说的是对还是错。”[注]我们当然希望“点”、“线”这样的术语指代的是我们平时谈论的点和线，但是希尔伯特认为任何涉及这些术语的证明，都只应通过公理推导而出，不该源于我们对这些文字的直观理解。

希尔伯特在几何学方面的成功自然而然地带来了一个问题：公理化方法是否能够应用到整个数学学科？是否能够找出一组公理，构建出数学学科的全部？包括希尔伯特和罗素在内的一些数学家认为这种公理化方法是可行的。但是，在讨论罗素、希尔伯特和其他人的研究前，我们还需要了解一下数学逻辑的发展。

布尔逻辑

逻辑一直是数学的组成部分。实际上，《几何原本》之所以如此有影响力（特别是在教育领域），欧几里得在写作时使用的方法功不可没。这本书不只是数学结论的罗列，更重要的是它向我们展示了如何从更简单的结论中得到这些结论。它不仅教你数学，还教你如何使用逻辑论证。为了真正受到教育，你必须熟悉这些论证。亚伯拉罕·林肯（Abraham Lincoln）曾经写道："最后我说，林肯，如果你不理解证明意味着什么，你永远当不了律师；我放下自己在斯普林菲尔德的工作，回到父亲家，待在那里，直到我只看一眼就能给出欧几里得6本书中任意命题的证明。那之后，我发现了证明意味着什么，于是继续我的法律研究。"注

逻辑一直用于数学论证，19世纪时成为数学研究的一个独立领域。乔治·布尔（George Boole）最早提出了这个方法，意识到代数可以应用到逻辑中。使用"与"、"或"、"非"等连接符陈述的某些逻辑推论可以简化为对符号的代数运算。

1847年，布尔首次发表了自己的结论，他将这些结论继续雕琢，最终写入了自己的著作《思维规律的研究：构成逻辑和概率的数学原理》（*An Investigation of the Laws of Thought: on Which are Founded the Mathematical Theories of Logic and Probabilities*），直到1854年才发表。

其他数学家将布尔的结论发展壮大，形成了如今人们所知的布尔代数。这种代数对我们相当重要，因为它既是数学逻辑的开端，也是计算机设计的基础（在注释中，我们给出了一个使用布尔代数的例子，证明"今天是星期六或今天下雨了"不正确，等价于"今天不是星期六并且今天没下雨"是正确的注）。

数学逻辑

布尔代数只能处理现在所谓的命题逻辑或零阶逻辑，并不包括"全部"或"部分"这类量词。像"一些正整数是质数"、"所有的整数都是有理数"这样的陈述，布尔代数无法处理。

下一步就是扩展布尔代数来包含量词并最终将数学所需逻辑完全转换为代数形式，即所有逻辑论证都可以用符号化操作表示。美国的查尔斯·桑德斯·皮尔士（Charles Sanders Peirce）和德国的戈特洛布·弗雷格（Gottlob Frege）分别独立完成了这项工作。弗雷格首先发表了关于量词的结论，但是皮尔士的研究对数学逻辑的后续发展起到了更为重要的作用。部分原因是皮尔士采用了更高级的标记，主要原因则是另一位德国数学家恩斯特·施罗德（Ernst Schröder）的研究，他撰写了一本关于数学逻辑的教科

书，这本书影响深远，解释并拓展了布尔和皮尔士的研究。注

一旦证明逻辑具备数学结构，就会导致一个问题：是否可以制造机器来进行逻辑计算？在继续对数学基础的讨论前，我们将谈一些题外话，介绍一些制造机器的人。

逻辑机器

英国经济学家、逻辑学家威廉·斯坦利·杰文斯（William Stanley Jevons）是最早意识到机器可以处理布尔逻辑的学者之一。他设计出逻辑钢琴（Logic Piano）来计算小真值表。逻辑钢琴像钢琴一样有键盘，用于输入初始假设。它于1869年制成，1870年在英国皇家学会展览。注逻辑钢琴并不是很实用，因为它只能处理容易由人工完成的简单问题，但是它却证明了机器可以处理逻辑。

1882年，皮尔士的学生艾伦·马昆德（Allan Marquand）在普林斯顿大学教授逻辑和拉丁语时，设计了自己的逻辑机器。这台机器类似杰文斯的机器，能够完成的任务相当有限。皮尔士致信马昆德，建议使用电机机械制造更高级的机器。但是，当时的普林斯顿大学校长认为皮尔士研究逻辑的数学方法是"异教徒的、非加尔文主义的"，因此马昆德应该停止教授逻辑，改为教授艺术史。马昆德成为一位成功的艺术史专家，但是他在逻辑学方面的研究就此止步。

大约在同一时期，皮尔士迎娶了自己的第二任妻子，人们发现皮尔士夫妇在婚前已经开始同居。绯闻终结了他的学术生涯。他被约翰·霍普金斯大学解聘，没有大学愿意雇用他。对于一个被罗素誉为"19世纪晚期最具独创性思想，当之无愧的最伟大的美国思想家"的学者而言，这不能不说是一个悲惨的结局。

保卫数学基础

弗雷格的研究引起了罗素的兴趣。罗素对弗雷格如何发展逻辑并不在意，而是对他将逻辑与数学联系在一起的观点感兴趣。弗雷格是"逻辑主义"（逻辑构成了数学的基础）这个概念的发起人之一。根据这个概念，人们首先发展出了逻辑，随后由逻辑构建了数学。一切都是根据准确规则建立的。弗雷格投入多年时间发展这个理论，但不幸的是，在即将发表第二卷研究的内容前——他在里面详细阐释了这个理论，他接到一封来自罗素的信，信中指出这个理论存在一个基本缺陷。弗雷格从集合出发，发展了自己的理论，他将集合视作聚集。他并未意识到，这种看似简单的想法

可能会导致问题。

罗素认为，集合应该被更认真地定义。弗雷格的构想让我们能够考虑所有可能集合的集合——最大可能集合，这产生了悖论。罗素联系弗雷格，解释了这个问题。弗雷格匆忙为自己的作品增加了附录："书稿完成后，自己研究的一个基础被动摇，或许很难有比这更不幸的事情降临在一个科学作者身上。这就是我在收到伯兰特·罗素先生的信后面临的问题，当时这一卷的印刷工作即将完成。"但是，尽管罗素实际上驳倒了弗雷格的研究，他依旧认为弗雷格的方法是正确的。弗雷格的研究可以重新来做。

这是他在自己最重要的数学研究——与阿尔弗雷德·诺斯·怀特海（Alfred North Whitehead）合著的《数学原理》（*Principia Mathematica*）中所做的事情。与弗雷格的方法一样，他们以逻辑为起点，从逻辑来到了算术。但是，进展相当缓慢。在完成三卷著作后（分别于1910年、1912年和1913年发表），他们只研究到实数。为了确保没有涉及无根据的假设，直到第362页（第一卷的中段）才有这样的陈述："从这个命题开始，直到算术加法被定义，才有1 + 1 = 2。"这一卷并未定义算术加法，它出现在第二卷。

读者也惊讶于这项研究的时间跨度。他们已经撰写了三卷，并且远未接近终点。下一卷或许会关于几何学，但是罗素和怀特海看到了巨大的工程量，决定放弃这个项目。

这是一项洋洋洒洒近2 000页的研究。有些页只有符号，但也附上了翔实的描述来介绍使用这些符号在做什么及原因——罗素是一位杰出的作者。尽管这个项目并未完成，已经出版的研究成果仍然影响深远，且影响范围不局限于数学界。它清晰的文字表达吸引了更加广泛的读者。T·S·艾略特（T. S. Eliot）对这一著作颇为赞赏，评论道："逻辑学家必须利用好英语，使这种语言令读者容易准确清晰地思考。或许，《数学原理》对语言的贡献要超过它对数学的贡献。"[注]

希尔伯特的方法

除了数学逻辑，19世纪还见证了非欧几何的诞生。除了关于平行线的公理，其他几何结构都是使用欧几里得几何构建的。不同公理在这些几何结构中得到证明，这些几何结构与对应的欧几里得几何结构大不相同。有人对这些公理系统最终会导致悖论持怀疑态度。给定一组连续的公理，我们能否确定无法证明一个陈述正确与否？非欧几何公理是否会导致悖论？我们能否确定欧几里得几何不包含矛盾？为了尝试回答类似的问题，数学中一个领域的模型需要在另一个领域中得出。

希尔伯特证明，如果算术是连续的，那么欧几里得几何也是连续的。有学者证明，如果欧几里得几何是连续的，那么非欧几何也是连续的。这些相关的连续性证明令人欣慰，但是人们意识到，连续性证明不应假设另一个领域是连续的。我们需要从一个能够被证明为连续的领域开始。因此，希尔伯特列出了他认为数学的公理基础应该具备的基本性质。

第一个性质是公理应该是连续的。公理不应该导致悖论——没有论述可以被证明是既对又错的。除此以外，应该能够证明这些公理是连续的。证明不应该涉及无法通过这些公理证明的概念。它应该能够由这些公理证明得到——从系统内得到。

第二个性质是公理应该是完整的。数学中的每个论述应该能够通过公理证明或反证。

“数学应该建立在一个完整、连续的公理系统上，并且在系统内进行一致性证明”的想法，就是后来所谓的希尔伯特计划（Hilbert’s Program）。1920年，希尔伯特提出了最初的计划。1928年，他增加了判定问题。他认为应该存在一个决策程序，能够告诉我们一个论述能否通过这些公理证明。希尔伯特所指的决策程序是一种清晰的计算过程，也就是我们现在所说的算法。[注]他认为，向这个过程输入公理和可能结论，它应该告诉我们这个可能结论是否可以通过这些公理证明。希尔伯特确信，这样的算法一定存在。判定问题就是寻找算法的问题。

尽管希尔伯特喜欢用这种有序、机械的方法研究数学，其他人并不是这样。英国数学家G·H·哈代（G. H. Hardy）确信希尔伯特是错误的。1928年，他在谈到判定问题时指出：“当然不存在这样的公理，我们应该感到庆幸，因为如果我们找到了一套机械的规则，为所有数学问题提供解决方案，那么数学家的活动就将结束。”[注]

希尔伯特并不赞同哈代的意见。他坚信这种算法是存在的，坚信数学中的所有问题都可以被解决，坚信在所有情况下都存在能够给出解决方案的算法。他曾经这样说道：“我们必须知晓。我们将会知晓。”他辞世后，这两句话成为他的墓志铭。他确信这个计划可以构建一个安全的框架，数学的所有方面都可以建立在这个框架下。

然而，事与愿违。在他去世之前，他的计划就被证明是无法达成的。首先有了哥德尔的结论，随后是邱奇和图灵的结论。

哥德尔结论

1931年，奥地利逻辑学家库尔特·哥德尔发表了一篇题为“论《数学原理》及相关体系中的形式上不可判定命题”（*On Formally Undecidable Propositions of“Principia Mathematica” and Related Systems*）的论文。在这篇论文中，他讨论了公理系统，这些系统足够强大，可以证明关于数字的结论——罗素和怀特海的《数学原理》就是这种系统的雏形。哥德尔证明了如果这些公理是连续的，那么它们就不是完整的。存在这些公理无法证明或反证的论述。他同样证明了无法在这个系统内证明公理的连续性。

哥德尔彻底摧毁了希尔伯特在1920年创立的项目。但是，判定问题仍然悬而未决。

图灵的结论

在同一时期，图灵开始了在剑桥的研究生涯。1935年春天，他参加了马克斯·纽曼（Max Newman）的数学基础课程。在课堂上，纽曼对哥德尔的结论进行了解释，图灵也了解到了判定问题。他下定决心证明希尔伯特是错误的。希尔伯特渴望构建通用算法。他的命题中内嵌了隐性的假设——这种算法是真正存在的，而图灵的计划则是证明这一假设本身就是伪命题。他罗列了一些远远超出算法能力范畴的问题。他做出的证明也支持了哈代的观点——没有哪些规则能够给全部的数学问题带来解决方案，因此数学家的研究是永远不会结束的。

希尔伯特并没有正式定义过所谓的决策过程。因为在他看来这可能并无必要。纵观历史，人们脑海中的数学等同于计算——数学这一术语也出现在天文学和占星学中，用于计算恒星、行星的位置。注 到19世纪，论证过程基本仍存在于计算范畴。大多数数学家或参与到算法开发研究中，或使用明确的算法——对算法的使用是做数学这门学问必不可少的部分。然而，像希尔伯特的判定问题这样的难题却让一些数学家和逻辑学家不禁要问：计算究竟是什么？算法的定义又是什么？

如果图灵想要证明某些特定的算法并不存在，那么他就必须回答这些问题。包括逻辑学家哥德尔、阿隆佐·邱奇在内的大师们都给出了自己的答案，但是图灵的解答却别具一格，他选择用设计理论计算机器的方式定义算法。稍后我们将看到各种完成计算的方式，但我认为，图灵采用的方法是最优雅、最简洁的。

在定义基本术语后，图灵论证过程的第二步是把自己的理论机器转化成一串串数字。也就是将一些由高级编程语言编写的程序转化成一系列二进制数字串、机器代码，让真正的计算机能够处理。

众所周知，今天的计算机使用二进制字符串，我们通过普通的语言与它们交互。很显然，肯定存在某种方式能够将我们的指令转化为二进制数字。但是在20世纪30年代，把指令转化成二进制数字的想法却是绝对的天方夜谭。要知道在这一时期，“Computer”所指的还是那些进行计算任务的人，人类计算人员收到的指令想必也都是一些日常所用的英语，而绝非一串0、1码。

当图灵定义好这些由一串串数字构成的算法后，就可以描述通用计算机。这种计算机能够将算法、数据作为输入，然后将数据套用到算法上。这是一个非常重要的理念。通用图灵机可以模拟任何一台图灵机。因此，通用图灵机能够完成其他图灵机能够完成的全部任务——运行所有的算法。

所有的现代计算机都是存储程序式计算机——数据和程序以完全相同的方式得到处理。这一想法实际上也源自图灵。当我们追随图灵的论证逐步深入时，我们就是在做计算，我们也就是计算机。在某一时刻，我们会突然意识到该如何将一串数字分解成两部分：一部分是算法，另一部分是将由算法处理的数据。实际上，从我们能够听从别人的指令完成任务开始，我们就已经变成了通用计算“机”。我们发现人类不仅是通用计算机，人类的计算能力能够媲美现有甚至未来可能出现的最强大的超级计算机器。

通用计算机显然非常重要，但是图灵将更多的注意力放到了将算法编码成数字串的过程中。他希望证明一些看起来并不难处理的问题，实际上完全超出了任何一台计算机的能力范畴。为了证明这一点，图灵借鉴了哥德尔的论断，后者则是从康托尔的对角论证法中获得启发。

图灵用来反驳那个著名的判定问题的方法是一段绝美的论证，有着很多环环相套的想法。我们将仔细研读这些论证，理解其中的每一个步骤。

我们将站在两个不同的角度评价图灵的这篇论文。一方面，这是一篇关于数学逻辑基础的论文，成功证明了判定问题的根本缺陷，从而终结希尔伯特计划。另一方面，正是这篇论文按下了计算理论以及计算机科学研究的启动键。这是一个旧时代的终结，也是一个新时代的开始。第一种审视方式着重强调了逻辑，第二种方式则强调了程序运行。在撰写这篇论文的时候，图灵显然采用了第一种方式，当这些理论被搬上讲台时，将它们与当今的计算机联系起来就更有意义，因此我们也倾向于以第二种方式看待这篇论文。

虽然我反复强调图灵论文如何优雅，但实际上这仍然是一篇相当难阅读的论文。学生们看到的往往是图灵想法的当代简化版。1958年，马丁·戴维斯（Martin Davis）根据自己在伊利诺伊大学教授的课程出版了《可计算性与不可解性》（*Computability and Unsolvability*）一书。1967年，麻省

理工学院人工智能实验室联合创始人马文·明斯基根据当时在麻省理工学院教授的一门课程，出版了《计算：有限和无限机器》（*Computation:Finite and Infinite Machines*）一书。这两本书有着极大的影响力，它们以一种更简单易懂的方式呈现了图灵的想法，它们让计算理论的研究成为计算机科学的一部分，也让计算机科学发展成了一个独立的学科。很多现代的方法都与这两位计算机科学家有关。例如，在解释图灵的判定问题之前，通常都会介绍更易理解的停机问题，而这一问题最早出现在戴维斯的书中。在本书中，我们也将以这种现代的方式解析图灵的著作。

新的理解方式并没有改变图灵的基本思路，只是着眼点不同。比起计算机能够计算怎样的数字，我们这些生活在现代的人可能对计算机能够运行怎样的程序更感兴趣。我们希望提出问题，看看是否有算法或者程序能够解答这些问题。图灵对判定问题的证明也带来了一个问题：一些特定的判定问题是不可判定的。

非确定性是图灵论文中的主要观点，也是研究他的想法的一个绝佳的着眼点。在下一章中，我们将看到一些不可判定的判定问题的例子，更清楚地了解这一术语的真正含义。

---

1. 伯特兰·罗素，《神秘主义和逻辑：和其他论文》（*Mysticism and Logic: And Other Essays*）第60页。

2. 我们将交替使用假设和公理。欧几里得确实区分了这两个术语。他的切入点是一组公理和另一组假设。现在我们可以整合这两个集合，并且统称公理。

3. 伯特兰·罗素，《数学原理的近期研究》第62页。

4. 约翰·亨利·凯查姆（John Henry Ketcham），《亚伯拉罕·林肯的一生》（*The Life of Abraham Lincoln*）第62页。

5. 我们来简单看看布尔代数如何工作。
   真值表如下：

$$T \wedge T = T, T \wedge F = F, F \wedge T = F, F \wedge F = F$$

$$T \vee T = T, T \vee F = T, F \vee T = T, F \vee F = F$$

   通常我们认为T是1，F是0。布尔代数给出了一种使用代数进行逻辑运算的方法。例如，“与”可以表示为乘法“×”。表述$X \wedge Y$就变成了$X \times Y$。其对应的真值表是：

$1 \times 1 = 1, 1 \times 0 = 0, 0 \times 1 = 0, 0 \times 0 = 0$

对于“或”，表述$X \vee Y$可以写作$X + Y - X \times Y$。

陈述P的否定可以表示为$\neg P$。在布尔代数中，$\neg X$可以表示为$1-X$，也就是说，如果$X = 1$，那么$\neg X = 0$。如果$X = 0$，那么$\neg X = 1$。

有了这些准备后，我们就有可能使用代数推导出基本的逻辑等式。我们将给出一个例子：$\neg(P \vee Q)$意味着$(\neg P) \wedge (\neg Q)$（这是德·摩根定律的一条）。我们将使用代数证明$\neg(X \vee Y)$与$\neg X \wedge \neg Y$的等价性。

注意$\neg(X \wedge Y)$对应$1 -（X + Y - X \times Y）$，而$\neg X \wedge \neg Y$对应$（1 - X）\times（1 - Y）$。简单的代数转换说明了这两个表达式都等于$1 - X - Y + X \times Y$。所以，这两个表达式是等价的。

6. 正如美国哲学家、数学家和计算机学家希拉里·普特南（Hilary Putnam）在《逻辑学家皮尔士》一文中提到的：

> 尽管据我所知，除了弗雷格没有人发表过关于弗雷格标记的论文，许多著名逻辑学家采纳了皮尔士–施罗德标记，许多著名结论和系统都与它相关。
>
> ……
>
> 弗雷格确实“发现”了量词，从这种意义上来讲，他确实有权主张自己的发现。但是，皮尔士和他的学生从更有效的意义上发现了量词。事实上，在罗素认可弗雷格的研究之前，弗雷格一直相对默默无闻。皮尔士似乎在世界逻辑学界更为著名。有多少认为“弗雷格发明了逻辑”的人了解这些事实呢？

7. 这台机器现在由牛津大学历史科学博物馆收藏。

8. 艾略特（T.S. Eliot），《每月标准》评论栏目，1927年10月，第291页。

9. 算法是一步接一步执行计算的过程。“算法”这个词起源于公元850年，波斯数学家花剌子模撰写了《关于印度数字的计算》一书。这本书还介绍了标准十进制数字表示法。尽管算法是以这位学者的名字命名的，但这个概念一直是数学的一部分。欧几里得的《几何原本》中包含了欧几里得算法——寻找两个正整数的最大公约数，但是早在公元前1600年，巴比伦人就发明了计算平方根的方法。

10. 安德鲁·霍奇斯，《艾伦·图灵：恩尼格码》，第120页。

11. 圣奥古斯丁警告基督徒应小心数理学派。他担心的是占星师，而非我

们现在所谓的数学家。

第二章 一些不可判定的判定问题

在本章中，我们将探讨三个不可判定的判定问题，每个问题都有相当的知名度。第一个问题是希尔伯特的“第10个问题”，第二个问题是波斯特的对应问题，第三个是停机问题。我们先准确描述一下什么样的问题才能成为判定问题。

我们将从一个简单的例子开始。考虑一下这个问题：给定两个正整数x和y，$x^2 + y^2$是一个正整数的平方吗？这个问题的答案取决于x和y的取值。如果我们选择$x = 3$，$y = 4$，那么$x^2 + y^2 = 9 + 16 = 25$，是5的平方。在这种情况下，这个问题的答案是“是的”。如果我们输入$x = 1$，$y = 1$，那么$^2x + y^2 = 2$，我们知道2不是某个整数的平方。所以在这种情况下，答案是“不是”。给定输入参数的特定值，问题就有明确的答案。这就是判定问题的本质，即“是否为判定问题”取决于输入参数。一旦输入参数确定，它就变成一个是非问题。

如果能够设计一个算法或计算机程序，在所有情况下都能给出正确的答案，那么这个问题是可判定的。如果无法设计一个在所有情况下都能给出正确答案的算法，那么这个问题是不可判定的。

我们上面提到的“$x^2 + y^2$是否是某个整数的平方”是一个可判定问题。我们很容易就可以写出一个程序，输入x和y，让它告诉我们，$x^2 + y^2$是不是一个整数的平方。

证明一个判定问题是不可判定的，这个过程通常十分困难。只埋头研究不足以设计出能够回答这个问题的算法。为了证明一个问题是不可判定的，你必须证明不存在能够解决这个问题的算法。证明某个东西不存在通常要比证明某个东西存在困难。证明某个东西存在的直接方法是找到一个你需要的具体实例，但是这离证明“不存在”还有十万八千里。为了证明某个东西不存在，你必须证明不可能找到这样的实例。

在下文中，我们将描述已知的不可判定的问题。我们将详细描述并讨论为什么它们是不可判定的。在后面的章节中，我们将深入研究蕴含在不可判定性证明之中的思想。

首先，让我们来看看波斯特的对应问题。在此之前，我们先介绍一下这位理论计算界的巨擘。

埃米尔·波斯特

1920年，埃米尔·波斯特（Emil Post）从哥伦比亚大学毕业，拿到了自己的博士学位。在博士论文中，他研究了罗素和怀特海的《数学原理》，证明了奠定零阶逻辑的公理——命题演算同时具备一致性和完整性。毕业后，波斯特来到普林斯顿大学，进行了为期一年的博士后研究。他的目标是研究《数学原理》中涉及的全部公理，并证明这些公理的一致性和完整性。

在研究过程中，波斯特采用了一种自创方法——标签系统（我们将在后面介绍这种方法）。他证明了《数学原理》中的公理在理论上可以简化为一种更简单的形式，类似于他的标签系统。他意识到，对《数学原理》中的公理的证明过程可以简化为对符号字符串进行的简单操作。

他的研究具有高度的原创性和突破性。他发现了构成哥德尔的不完备性定理和判定问题反证基础的主要概念。1921年时波斯特就在研究这一领域的问题。哥德尔在1931年发表了自己的研究成果，邱奇和图灵在1936年发表了他们的研究成果。因此，波斯特领先他们10年。不幸的是，当时他并未发表任何研究成果。波斯特患有双相情感障碍，1921年，他第一次发病。在此后的生命中，他必须定期住院，接受电击治疗——这是当时的标准疗法。[注]在第一次住院治疗后，他与医生一起研究病情，探讨自己能否免受未来狂躁情绪的影响。

他限制了自己研究的工作量，每天准备两个问题。如果一个问题让他太过兴奋，还可以研究另一个。他的妻子和家庭给了他莫大的支持。但是即便如此，直到1930年他还是没有发表任何论述。

1935年，波斯特入职纽约城市学院，并在这里一直工作到离世。在这段时间中，他的早期工作变得家喻户晓，他也继续保持高产和原创。芝加哥大学数学家罗伯特·索尔（Robert Soare）写道："1939年，图灵提出纯计算性理论的主题。波斯特继续了之前的传统，用简单、直观的方式探索最基本的概念，比如计算性和可计算枚举集合，以及相关计算性和图灵可归约性。从1940年到1954年波斯特辞世，15年来他在这一领域的贡献可谓举足轻重。"

1946年，波斯特发表了他的对应性问题。这是一类非常容易描述的不可判定问题，我们将以此为切入点。

波斯特的对应问题

想象一个目录列出了很多种"瓷砖"。目录中的每块瓷砖都包含两个由1和2组成的序列，其中一个序列在另一个序列下面。你可以随意订购任意数量

的各种瓷砖。最终目标是找出一种瓷砖组合——当这些瓷砖排成一排的时候，顶部所有数字组成的序列与底部所有数字组成的序列相等。为了说明这个问题，我们来看两个例子。

第一个目录

第一个目录有如下三种类型的瓷砖（如图2–1所示）。

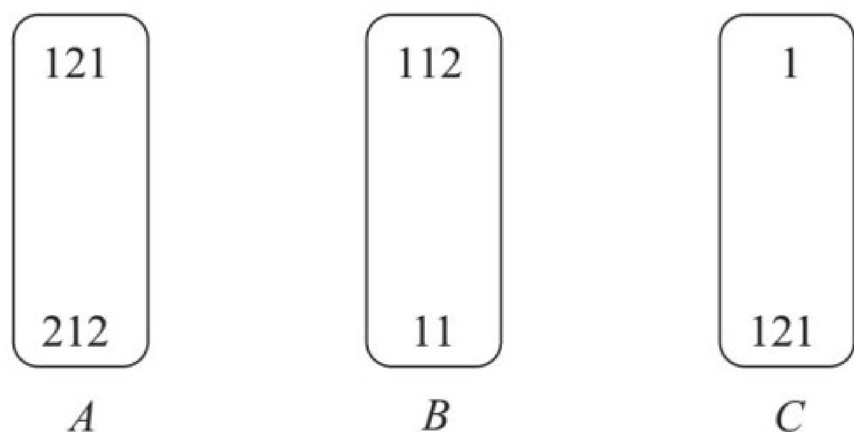


图2–1 第一个目录的三种瓷砖

我们可以任意选择各种类型的瓷砖，不限数量（包括0，但是至少要选择一种类型的瓷砖）。我们将这些瓷砖排成一排，以检查顶部序列是否等于底部序列。

在这种情况下，选择4块瓷砖就可以解决这个问题：一块A、一块C和两块B，并且按照$BABC$的顺序来摆放（如图2–2所示）。

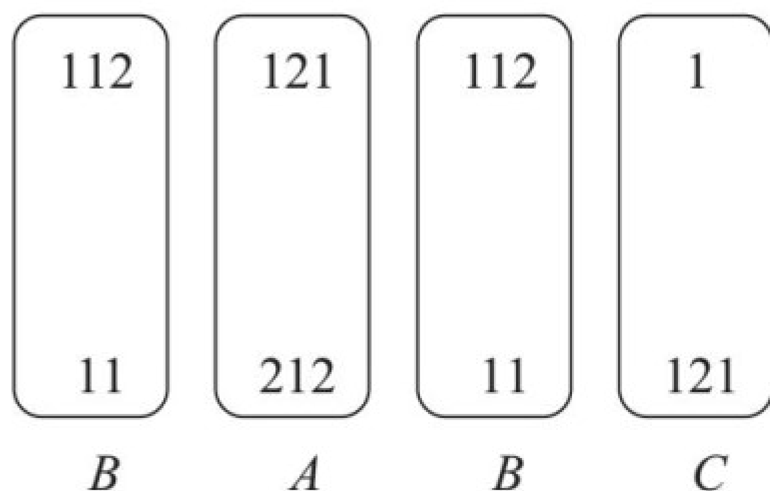


图2–2 第一个目录的4块瓷砖

按照这个顺序，我们得到了1121211121这个序列，而且顶部和底部的序列一致。所以，我们找出了这个目录的解决方案。

现在，我们来看一下另一个没有解决方案的目录。

第二个目录

还是有三种类型的瓷砖（如图2–3所示）。

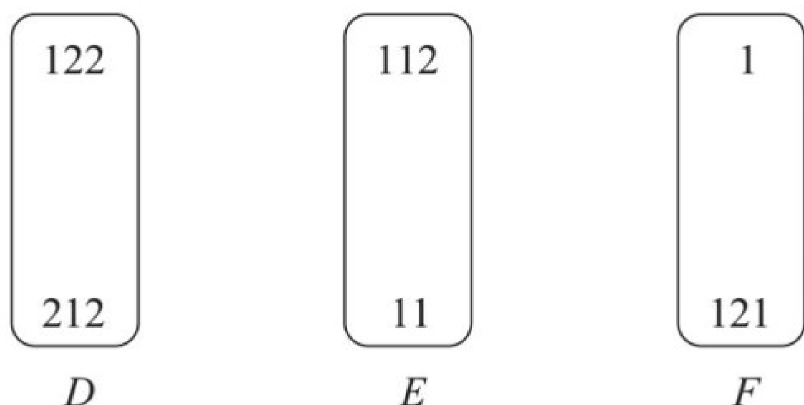


图2–3 第二个目录的三种瓷砖

我们要尝试设计一个算法，认真执行所有可能的步骤。我们将从最左侧的瓷砖开始，向右依次增加瓷砖。

很明显，我们不能将瓷砖D放在最左侧，因为如果我们这样做，顶部的字符串会以1开始，而底部的字符串会以2开始。

如果将E作为第一块瓷砖，让我们来看看会发生什么。在这种情况下，顶部字符串以112开始，底部字符串以11开始。为了找到解决方案，我们需要一个底部以2开始的瓷砖。因此，下一步我们必须选D。来看ED，我们将在顶部得到112122，底部得到11212。我们需要一块底部字符串以2开始的瓷砖作为下一块瓷砖。因此我们必须继续选择D，这样我们就得到了EDD。顶部字符串为112122122，底部字符串为11212212。我们还需要一块底部字符串以2开始的瓷砖。另一个D被添加到右侧——这个过程会一直重复下去。在每个阶段，我们都必须向右侧增加D，但是我们永远找不到解决方案，因为顶部字符串总比底部字符串长一点。这告诉我们，不存在以E开始的解决方案。

我们已经看到，不存在以D或E开始的解决方案。唯一的可能性是以F开始。在这种情况下，顶部的字符串以1开始，底部的字符串以121开始。为了找到解决方案，我们需要一块顶部字符串以2开始的瓷砖。很不幸，不存在这样的瓷砖。因此，没有以F开始的解决方案。我们已经考虑了所有可能性，所以我们知道这种目录不存在解决方案。

给定任意瓷砖目录，有两种可能性：要么存在一种解决方案，要么不存在解决方案。给定一个目录，我们希望找到一个过程，告诉我们这个问题是否存在解决方案。波斯特证明了这种判定问题是不可判定的。这意味着不存在算法或等价的计算机程序，可以将这些瓷砖目录作为输入，告诉我们每个目录是否存在解决方案。这是一个令人震惊的结论！需要明确的是，这并不是说没有人发现能够奏效的算法，而是不存在这样的算法。

我们将深入挖掘这一点，并通过设计一个“看似能够奏效”的算法来研究它的影响。

一个算法

这里有一个算法，我们称其为算法I：给定含有有限瓷砖数的任意目录，首先查看是否存在一块瓷砖是对应问题的解决方案。如果存在，我们就不用再做其他工作，并且得到了解决方案；如果没有一块瓷砖能够解决这个问题，尝试使用两块瓷砖可能的解决方案。如果有解决方案，我们将找到

它。如果没有两块瓷砖的解决方案，尝试所有可能的三块瓷砖的组合，以此类推。

如果我们在第一个目录上尝试这个算法，我们首先会看A、B或C是否能够给出解决方案。因为没有一块瓷砖能够给出解决方案，我们会检查AA、AB、AC、BA、BB、BC、CA、CB或CC是否能够给出解决方案。因为没有解决方案，我们将查看三块瓷砖的组合（这里有$3^3 = 27$种可能）。再次没有解决方案后，我们来看4块瓷砖的组合，这里有3^4种可能组合。我们会在这次迭代中找到解决方案。

很明显，我们的算法不够高效，运行起来相当缓慢。举个例子，如果A不是解决方案，那么AA或AAA也不会是解决方案；如果能够排除这些组合，就可以提高运算速度。但是，我们的目标是给出一个容易描述的简单算法，效率或运行速度并不是我们关心的问题。我们唯一的兴趣在于这个算法如何实现，是否能够在有限时间内停止。

尽管很消耗时间，如果解决方案存在，算法I最终能够找出它。因此，算法I能够在所有存在解决方案的对应问题中奏效。如果我们给出的目录不存在解决方案，算法I不会奏效——它会一直运转下去。

是否存在一个不同的算法，能够告诉我们某个目录不存在解决方案？答案是没有。如果存在这样一个算法，我们称之为算法II，我们可以设计算法III，给定一个目录，让算法I和算法II同时运行。如果目录存在一个对应问题的解决方案，那么在有限时间后，算法I将告诉我们。如果目录不存在解决方案，那么算法II将在有限时间后告诉我们。这意味着算法III是一个“能够告诉我们目录是否存在解决方案”的算法，但是波斯特的结论证明并不存在这样的算法。因此算法II并不存在。

我们有必要深入挖掘一下这一点究竟意味着什么。因为我们已经注意到算法I相当缓慢，但是如果在一台高速计算机上运行这个算法，并且给它一段相当长的时间，结果又会怎样？如果在世界上最快的计算机上运行这个算法，我们输入第一个目录，它一瞬间就可以给出答案。给定任意目录，我们能将它输入这台计算机。我们可以让算法I运行一段相当长的时间（比如10年），如果它没有在这段时间内得出结果，我们是否能够得出“它会一直运行下去，因此我们输入的目录不存在解决方案”这个结论？答案还是不能。如果答案是可以，我们就得到了算法II。然而，前面已经提到，这个算法并不存在。从这个论证中，我们可以推断出，一定存在一个有解决方案的目录，但是即使在世界上最快的计算机上运行算法I10年，还是找不出解决方案。

从波斯特的结论中我们获知，存在没有解决方案的对应问题，或者说这等

同于没有算法能够判定这个判定问题。另外，我们推导出了一些令人震惊的事实：第一个事实是给定任意时间间隔（比如一个世纪）和任意一个试图解决对应问题的算法，一定存在一个有解决方案的目录，让这个程序在这个时间间隔内无法找出答案；第二个事实是没有解决方案的目录背后隐藏着真实问题。当答案是“是”的时候，一定存在算法能够正确回答判定问题，尽管这些算法可能耗费相当长的时间，但是当答案是“否”的时候，一定不存在能够给出正确答案的算法。

含有更多符号的对应问题

我们前面讨论的是有1和2的字符串。我们可以考虑一些拥有不止两个符号的情况。在这些情况下，因为增加符号不会使事情变得更简单，所以“判断瓷砖目录是否有对应问题的解决方案”仍然是不可判定的。在下面的例子中，我们使用了符号0、1、A、B和*（如图2–4所示）。

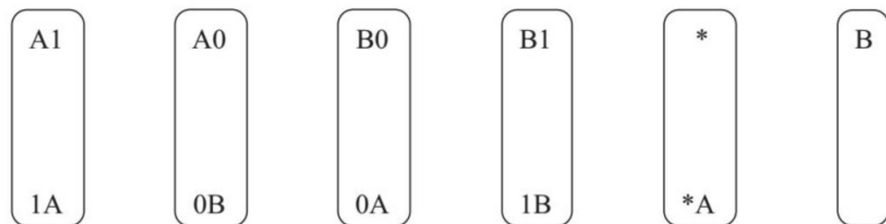


图2–4 多种符号的目录

这个例子将与下一章产生联系。它有很多解决方案。读者可以尝试自己找出一些。

你可能会提问：如果我们将状态数减少到1会发生什么？在这种情况下，问题就变成是否存在一种排列方案，让顶部序列的符号数等于底部序列的符号数。

这里有一个例子（如图2–5所示）。

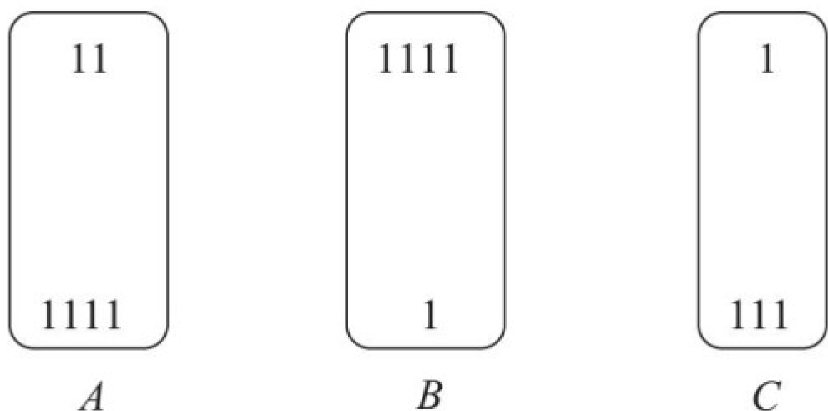


图2–5 存在解决方案的案例

当我们只有一个符号的时候，存在一种简单的处理方式。我们赋予每块瓷砖一个数字，这个数字是顶部符号数减去底部符号数（在本例中，A被赋值–2，B被赋值3，C被赋值–2）。如果存在瓷砖被赋值0，那么它本身就是一个解决方案。如果在目录中至少存在一个正数，一个负数，那么这个对应问题就存在解决方案。如果所有瓷砖的赋值都是正数，或者都是负数，就不存在解决方案。据此，我们得出了一个判断瓷砖目录是否存在解决方案的算法。如果我们将瓷砖限制为只有一个符号，对应问题就是可判定的。

在我们的例子中，有一块正数瓷砖和一块负数瓷砖。因此，一定存在解决方案。为了找到这个解决方案，我们先要找出等于0的组合。三个A和两个B分别得到–6和＋6。因此，$AAABB$是一个解决方案。

希尔伯特的第10个问题

1900年，希尔伯特列出23个他认为数学中尚未解决的最重要的问题。这些问题对20世纪上半叶的数学研究方向产生了至关重要的影响。第一个问题是康托尔连续性假设，之后我们会讨论。第二个问题是证明算术公理是连续的。正如我们提到过的，哥德尔证明了“无法在这个系统内证明公理的连续性”。第10个问题与丢番图方程有关。

丢番图方程是有整数系数的多项式方程。我们希望找到这个方程的整数解决方案。一个著名的例子就是毕达哥拉斯定理[注]：$x^2 + y^2 = z^2$。解决方案是毕达哥拉斯三元组，分别对应直角三角形的三条边。我们熟知的三

元组有两个：3、4、5和5、12、13。尽管$x^2 + y^2 = z^2$有很多整数解决方案，但$x^3 + y^3 = z^3$却没有正整数解决方案。所以，一些丢番图方程有解决方案，一些没有解决方案。[注]

希尔伯特的第10个问题本意是设计一个算法：将丢番图方程作为输入，告诉我们是否存在正整数解决方案。这是一个判定问题。如果你输入$x^2 + y^2 = z^2$，它会回答存在。如果你输入$x^3 + y^3 = z^3$，它会回答不存在。这再一次证明，这个问题是一个不可判定的判定问题。尤里·马季亚谢维奇（Yuri Matiyasevich）在1970年给出了完整的证明。该证明建立在马丁·戴维斯（Martin Davis）、希拉里·帕特南（Hilary Putnam）和朱莉娅·罗宾逊（Julia Robinson）的研究基础上。这个证明已经超出本书的范畴，但是让我们再次感受这个证明对于不可判定的问题有什么意义。

考虑一下这个算法：首先，数清丢番图方程中的变量个数。然后，将1带入所有变量，检查是否是解决方案。如果发现解决方案就停止，否则尝试将1和2的组合代入变量。如果没有发现解决方案，尝试代入1、2、3的组合，以此类推。

如果存在一个整数解决方案，我们的算法就会找到这个答案。对于$x^2 + y^2 = z^2$，在尝试从1到5所有可能的组合后就会发现答案。但是，如果我们把$x^3 + y^3 = z^3$输入这个算法，它会一直运行下去。因此，我们知道希尔伯特的第10个问题是不可判定的，而且如果存在解决方案，我们已经找出一个能够正确回答这个问题的算法。我们可以推导出，没有算法能够正确识别不存在正整数解决方案的丢番图方程。

停机问题

我们已经将所有内容输入一台计算机，并且看到计算机死机了。如果有算法能够告诉我们，一个程序将会在某个给定数据上死机，这会十分有帮助，这就是所谓的停机问题。给定一个程序和初始数据，判断这个程序是否会在这段数据上停止。这是另一个判定问题，并且它也是不可判定的。它与另外两个判定问题拥有相同的性质。当答案存在的时候，存在算法能够起效，但是当答案不存在的时候，没有算法能够起效。在这种情况下，这个算法只会让程序去运行这段数据。如果它停止了，就输出“是的”。[注]

剑桥的图灵

当然，图灵在撰写自己的论文时，并不知道任何不可判定的判定问题的例

子。图灵和邱奇最早理解了“哪些问题是不可判定的”。希尔伯特的判定问题是给定一个判定问题，找出在任意情况下都能正确回答这个问题的算法。希尔伯特假设这种算法是存在的，即所有判定问题都是可判定的。如果图灵能够证明一些问题是不可判定的，那么他就能证明希尔伯特研究判定问题的方法是存在根本缺陷的。他面临一个相当大的挑战。他需要找出一个可以证明为“不可判定”的非确定性判定问题。他的证明必须展示出没有算法可以解决这个问题。但是在当时，算法究竟意味着什么并没有一个明确的定义。所以，图灵的第一步就是给这个术语一个准确的定义。图灵在发明理论计算机器的时候完成了这项工作。

---

1. 毕达哥拉斯定理，即中国的勾股定理。——译者注

2. 这种治疗方法现在被称为电休克治疗，在通常情况下与肌肉松弛剂配合使用。波斯特的年代没有肌肉松弛剂。电击会导致全身抽搐。波斯特57岁时，在接受完生前最后一次电击治疗后不久，死于心脏病发作。

3. 实际上，当n大于2的时候，$x^n + y^n = z^n$不存在正整数解。这就是著名的费马大定理。费马在一本书的书页边缘写下了这些论述，并进行了杰出的论证。但是，因为边缘部分不够大，不足以写下全部内容。1994年，也就是357年后，安德鲁·怀尔斯（Andrew Wiles）最终证明了这一结论。大多数数学家并不相信费马进行了合理证明。这一结论更恰当的称呼应该是费马猜想或怀尔斯定理。

4. 我们研究的三个问题有一个相同的性质，即存在一个算法，在有的情况下能够回答答案为“是”的判定问题，但是无法回答答案为“不是”的判定问题。类似这样“可以正确回答‘是’的情况，但无法回答‘不是’的情况”的算法，有时被称为部分可判定的，而非可判定的。

第三章 有限自动机

自动机是计算机器的简单理论模型。在本书中，我们将探讨两种类型的自动机：有限自动机和图灵机。

如今，人们将有限自动机视作简化版图灵机。它们没有图灵机的计算能力，但能够进行一些不复杂的计算。它们更易于理解和使用，因此在研究更加复杂的模型前，我们有必要研究一下有限自动机。尽管有限自动机更简单，但在定义图灵机之后数年，这个概念才被提出。它们当时的定义与今天我们描述它们的方式截然不同。

沃伦·麦卡洛克（Warren McCulloch）当时正在研究脑部创伤，治疗精神病人。他想研究出一种解释大脑如何工作的理论。沃尔特·皮茨（Walter Pitts）最初被培养成为一位逻辑学学者，但是却在全新的数学生物物理学领域发表论文。两人于1942年前后结识，认识到他们对相同类型的问题感兴趣后开始联手研究，彼此取长补短。

他们发表的第一篇论文题为“神经活动中内在的思想逻辑演算”（*A Logical Calculus of Ideas Immanent in Nervous Activity* ）。在这篇论文中，他们借助细胞对神经元进行了建模。每个细胞都有多个输入，但只有一个输出。一个细胞的输出必须成为另一个细胞的输入。输入有两种类型：抑制的和兴奋的。如果兴奋的输入超过了一定阈值，并且没有抑制输入，细胞将会被激活。细胞的集合和它们之间的连接被两人称作神经网络。

麦卡洛克和皮茨意识到，这是大脑实际运作的简化模型，但是研究神经网络可以得知神经网络如何处理逻辑活动。他们的网络模型与神经元和人类大脑具备相同的基本特征，因此他们希望自己的研究能够揭示人类逻辑推理的奥秘。

他们的论文引起了约翰·冯·诺依曼和诺伯特·维纳（Norbert Wiener）的注意。两位学者对这篇论文印象深刻。美国著名的数学家、哲学家维纳看到了其中蕴含的力量。他意识到，这一观点具有广泛的适用性，可以发展出控制论”。[注]控制论将催生可以学习的机器的理念，反过来也会孕育人工智能。

冯·诺依曼认识到，麦卡洛克和皮茨对细胞和细胞间连接的描述，同样可以应用到电子组件和计算中。他在《关于EDVAC的报告草案》（*First draft of a report on the EDVAC*）一文中对此进行了详细描述，这篇论文我们将

在后面提到。正是冯·诺依曼的这篇论文，奠定了现代计算机构建[注]的基石。

另一个受到麦卡洛克和皮茨论文影响的人是马文·明斯基。1954年，明斯基在自己的博士论文中对神经网络进行了研究，展示了如何使用这些网络对自动机进行全面描述。明斯基的著作《计算：有限和无限机器》是这一领域的经典之作，高屋建瓴地描述了自动机和计算理论。通过对比物理学，明斯基在这本书的前言中解释了这种使用理论机器研究的理论为什么能够发挥作用：

> 与物理学使用统计定义事件的方法不同，我们使用逻辑定义的计算或表达式。它们被联系在一起，不是通过几何或能量性质，而是通过它们与类似机器或类似定义之间的关系。我们能够使用机器组件进行简单的交互，应用最显而易见的逻辑命题。面对等价的现实物理机器时，我们必须解决极端复杂的分析等式。

自动机被划分为两类：一类具有有限内存，另一类具有无限内存。我们首先研究有限的一类。

有限自动机

有限自动机是一个简单的理论计算模型。它包括有限数量的状态。状态分为两类：开始状态（每次计算的起点）和接纳状态。这个机器的输入包括一系列符号。

提到符号，我们通常指的是一个字母表，符号序列指的则是字符串（字母表的元素通常被称为字母，即使它们实际上是数字）。

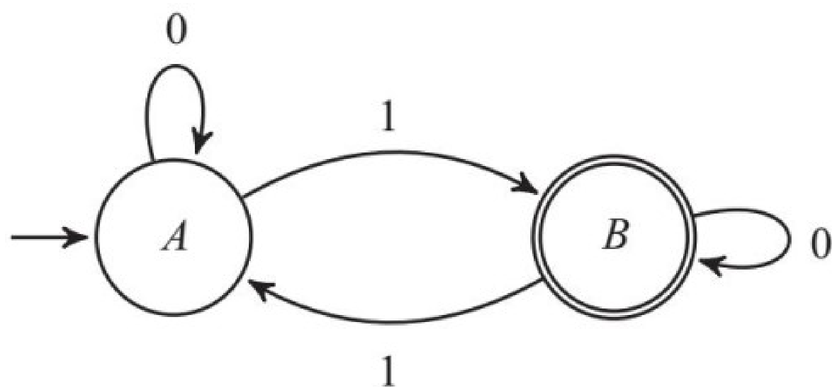
这个机器在开始状态启动，读入输入字符串上的第一个符号。有一系列规则告诉这个机器接下来应该如何移动。每条规则都与这个机器当前所处的状态以及将要读入的输入符号有关。在机器移动到新状态后，下一个输入符号被读入，机器找到对应规则，移动到指定状态，这个过程循环往复。当我们来到输入字符串的结尾时，计算就会停止。如果我们在接纳状态结束，输入字符串就会被接纳；否则，字符串就会被拒绝。

如果使用图表，前文所述的信息会更容易理解，在后文中我们也会使用这种方法进行解释。在这些图中，我们用圆圈表示状态，将状态的名称标记在圆圈中心。一个箭头指向开始状态，并且这个箭头不来自其他状态。接纳状态由双圆圈表示。连接状态的箭头上标记了输入符号，这些箭头将我

们从一个状态带到另一个状态。

我们的第一个机器

在图3–1中，我们展示了第一个有限自动机M_1。开始状态被标记为A，接纳状态被标记为B。离开A的两个箭头告诉我们，如果我们在状态A中，输入0，则继续留在A；输入1，则从状态A移动到状态B。如果我们在状态B中，输入0，则停留在状态B；输入1，则移动到状态A。机器M_1读入1就会切换状态，读入0则保持现有状态。



M_1：有奇数个1的字符串

图3–1 有限自动机

假设输入字符串是101。机器从开始状态A开始，读入第一个输入符号：1。它移动到状态B，读入第二个输入符号：0。它留在状态B，读入下一个符号：1。在读入这个符号后，它切换回状态A。它还想继续读入下一个符号，但是字符串已经结束，所以机器停止。因为我们在状态A结束，而A不是接纳状态，所以字符串101被M_1拒绝。

让我们用一个不同的输入字符串100重复这一个过程。机器从状态A开始，读入第一个输入符号：1。它移动到状态B，读入第二个输入符号：0。它留在状态B，读入下一个符号：0。它继续留在状态B。它还想继续读入下一个符号，但是字符串已经结束，所以机器停止。因为我们在状态B结束，并且这是接纳状态，所以字符串100被M_1接纳。

我们可以用语言来描述M_1会接纳哪种字符串，拒绝哪种字符串。值得一提的关键点在于，只有当我们读入1时，才会改变状态。由于只有两个状

态，我们从非接纳状态开始，读入偶数个1后，我们将处于状态A，读入奇数个1后，我们将处于状态B。因为B是唯一的接纳状态，所以我们知道，这个机器接纳含有奇数个1的字符串，拒绝含有偶数个1的字符串。

这个机器被当作检查输入字符串奇偶性的工具。在信息传输过程中，我们将奇偶性检查作为错误检测的一种简单方法。我们将在下一节中看到例子。

字母表和语言

正如我们看到的，字母表是一组符号。如果仅限于小写形式，英语中使用的标准拉丁字母表有26个符号。机器M_1的例子中使用的字母表仅包括0和1。因此，这是一个二元字母表。在上一章中，我们粗略地了解了对应问题。在一个例子中，我们使用的1和2组成了一个二元字母表。在另一个例子中，我们只使用了一个符号：1，这是一个一元字母表。

另一个曾经被广泛使用的字母表的例子是摩尔斯电码（Morse code）。有人说，在摩尔斯电码中，拉丁字母表中的每个字母由点和破折号表示。但是，这种说法不够准确。每个字母是由点、破折号和一个终端暂停表示的。在传递摩尔斯电码的时候，暂停与点和破折号同样重要。所以，摩尔斯电码的字母表是三元字母表，而非二元字母表。

正如我们在上一章中看到的，使用一元字母表的例子相当简单，但是当我们使用二元字母表时，事情就变得复杂。一般来说，能够用一元语言表示的东西相当有限，但是如果你在字母表中增加一个符号，让它变为二元，就能表达更多的东西。然而，单纯增加字母表的符号数，使其大于二元，并不能增加能够用它表示的内容的数量。这是因为对于任意数量的字母表，我们都有办法将每个字母转化为二元数字的字符串。因此，由任意字母表书写的任意字符串都可以重新编码为仅使用二元字母表的字符串。当然，如果我们正在使用一个包含多个字母的字母表，并且将其转化为使用更少字母的字母表，转化所得的字符串的长度将比原来的字符串长。

计算机科学中字母表一般是二元的，但是只要这个字母表至少包含两个字母，我们就可以使用它。实际上，一些早期计算机并不使用二进制，而是使用标准十进制。我们将会研究的早期计算机ENIAC，就是其中的一个例子。尽管二元字母表对计算机科学并不是必要的，但它足够应付各种任务。任何需要使用任意字母表处理的内容，都可以被重新编码为二进制字符串。我们之所以使用二进制，是因为二进制非常容易在机器上实现。两个字母可以用一个开关来表示：处于开启或关闭位置，也可以用电荷有无来表示。

美国标准信息交换代码（American Standard Code for Information Interchange，简称ASCII）是一种使用二进制字符串编码字母和控制符号的方法。每个字符被编码为由7个二进制数字组成的字符串。例如，A被编码为1000001，a被编码为1100001。因为在数据传输过程中，通常以比特为单位发送，数据字符串会添加一个奇偶校验位，以进行错误检测。

注 在发送前，这一位被添加到7位数据字符串的最后一位。奇偶校验位（1或0）用来保持字符串中1的总数为偶数。接收者会检查这个字符串，看奇偶性是否正确。如果不正确，证明字符串受到了破坏。类似的理念还被用于信用卡卡号——信用卡卡号的最后一位（有些信用卡卡号的第一位）被称为校验位，可以通过其他15位数字计算得出。当你使用信用卡时，这个校验位就会被检查，以确认信用卡卡号是否输入正确。注

正则语言

一个机器接纳的字符串集合被称为这个机器的语言。因此，M_1的语言是所有含有偶数个1的、由0和1组成的字符串。如果这个机器是有限自动机，那么它的语言就是正则语言。M_1接纳的含有偶数个1的、由0和1组成的字符串，就是我们接触的第一个正则语言。

有限自动机和回答问题

有限自动机会接纳或拒绝字符串。我们可以把这个过程想象为对一个问题做出“是”或“否”的回答。当它在接纳状态停止时，它在说“是”，当它在非接纳状态停止时，它在说“否”。实际上，人们经常将这个过程视作回答判定问题。机器M_1可以被视作在回答：这个由0和1组成的字符串里是否有奇数个1？请注意，这是一个判定问题。给定一个输入字符串，它就会变为是非问题。

从这一点看机器M_1，我们可以赋予它的两个状态实际意义。状态A对应机器读入了拥有偶数个1的字符串，而状态B对应机器读入了拥有奇数个1的字符串。

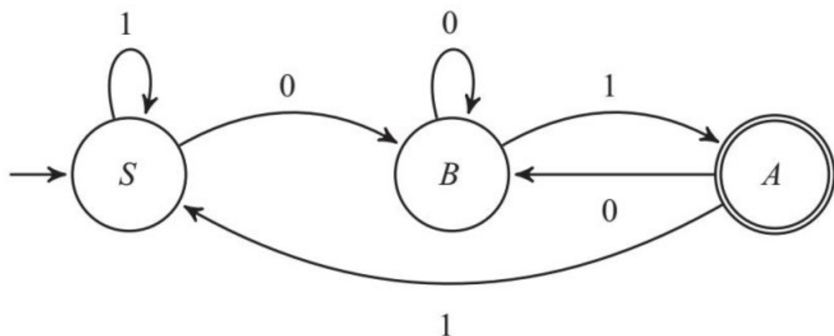
我们可以将有限自动机看作系统回答某些简单问题的方法。事实上，我们可以把有限自动机视作计算机程序或算法。我们会用一些例子来说明这个观点。

考虑一下判定问题：输入字符串是否以01结尾？我们将说明如何设计一个有限自动机来解决这个判定问题。

我们用S来表示开始状态（如图3–2所示）。如果一个字符串以01结尾，那么我们想要回答“是”，这等价于我们想在接纳状态停止。我们将用A来表示接纳状态。

当我们设计有限自动机的时候，一定要记住，一次只能读入一个输入符号。我们无法提前看到字符串的下一位，因此我们永远无法提前获知输入字符串是否要结束。当我们读入一个输入符号却没有读到任何输入的时候，才知道来到了字符串的结尾。

我们来考虑一下不同的情况。假设第一个输入符号是0。它有可能是我们想要接纳的输入字符串01。读入第一个0，意味着我们正走在通往接纳状态的路上。如果机器只读入了一个0，我们就用B来表示这个状态。如果我们在状态B，并且读入了一个1，那么我们就要移动到接纳状态A。



M_2：以 01 结尾的字符串

图3–2 用有限自动机解决判定问题

如果我们位于状态A，并且接收到了另一个输入符号，那么我们需要离开状态A。

总结一下：接纳状态A对应只读入了01，状态B对应只读入了0。状态S可以被视作开始状态，这个状态既不对应A，也不对应B，也就是只读入了一个1，但没有0。

一旦我们赋予状态具体含义，就很容易在读入输入符号时决定状态的转变。例如，如果我们在状态S，并且接收到一个1，我们就会停在那里；如果我们接收到一个0，我们就应该移动到状态B。如果我们停留在状态B，并且接收到一个0，我们就应该继续停留；如果我们接收到一个1，我们就应该移动到接纳状态。如果我们在接纳状态，并且接收到一个0，我们就应该移动到状态B；如果我们接收到一个1，就应该返回状态S。

图3–3描述了根据这些叙述产生的有限自动机。

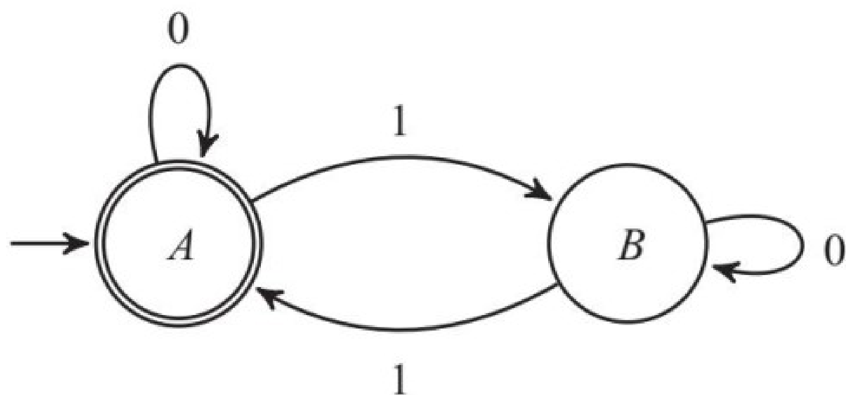


图3–3 根据输入符号决定状态转变

问题的否定

交换接纳状态和非接纳状态就可以回答机器M_1解决的问题的否定。“这个由0和1组成的字符串是否没有奇数个1”，即“这个由0和1组成的字符串是否有偶数个1”，图3–3中机器M_3回答了这一问题。这是M_1的接纳状态和非接纳状态互换的结果。

请注意，M_3接纳000，因为我们仍然停留在状态A。这是因为000包含0个1，而0是一个偶数。同理，M_3接纳00、0，甚至没有符号的空字符串（用ε来表示）。实际上，如果开始状态同样是接纳状态，机器就会接纳ε。这说明在输入任何符号前，机器已经处于接纳状态，所以它将接纳不包含任何符号的空字符串。

交换一个有限自动机的接纳和非接纳状态，可以回答原问题的否定。这告诉我们，正则语言的补集仍然是正则语言，即如果给定一个正则语言，不属于这个语言的字符串的集合仍然是正则语言。这个发现看起来微不足道，但结论本身很重要。当我们在后文中探讨图灵机的时候，我们将发现有时候一个图灵机语言的补集，不是其他任何图灵机的语言。实际上，这个性质构成了我们之后将要讨论的不可判定的判定问题。

忽略图表中的陷阱

一些传统方法可以帮助我们简化理论计算模型。其中一个最有用的方法就是忽略非接纳陷阱。我们将在下一章介绍图灵机，下一章的每个图表都是用这种方法绘制的。我们先举例说明如何使用这个方法。

考虑一下“输入字符串是否以01序列开始”这个问题。如果我们先后接收到一个0和一个1，我们想要接纳这个字符串。所以，在接收到01后，我们的机器一定处于接纳状态，并且无论后续接收到什么符号，都必须停留在这个状态。如果我们接收到的第一个符号是1，我们将进入拒绝状态，并且无论后续接收到什么符号，都一直停留在这个状态。图3–4是实现这一叙述的自动机。

这个自动机有两个状态，一旦进入这两个状态就无法离开。这些状态被称作陷阱。在M_4中，C和D都是陷阱。通常情况下，为了简化自动机的示意图，不是接纳状态的陷阱会被忽略。图3–5中的自动机与图3–4中的自动机是同一台自动机。在图3–4中，状态D是一个非接纳状态。在图3–5中，这个状态被忽略了。请注意，在图3–5中，没有来自A的箭头能够告诉我们，如果输入为1应该做什么。这是在假设如果我们处于状态A，并且输入了一个1，我们就会强制进入不被接纳的陷阱，但是通俗地说，如果我们处于状态A，输入了一个1，那么计算就“死掉”了。如果机器处于状态B，并且接收了一个0，那么计算也会“死掉”。

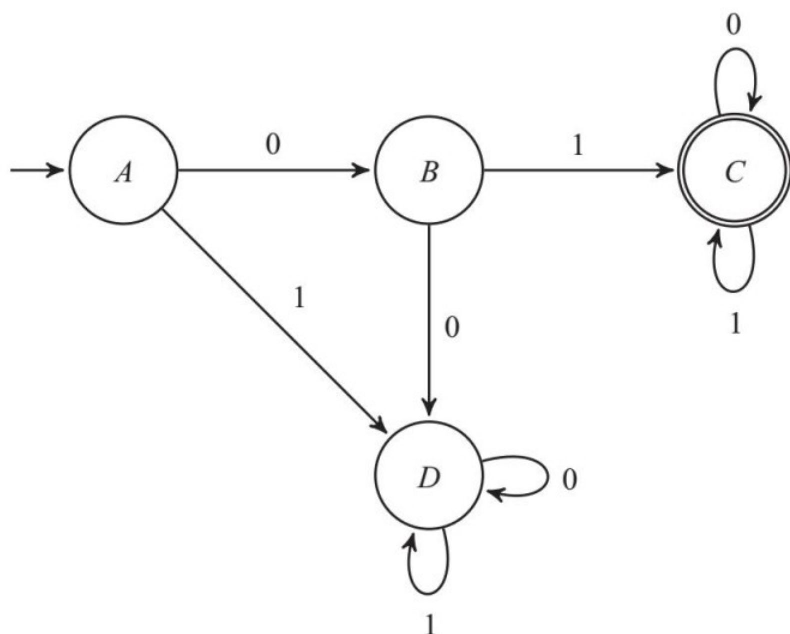


图3–4 以01序列开始的自动机

当然，即便我们从图中剔除了陷阱，还是有必要记住这些状态。为了设计有限自动机来回答否定问题“这是一个不以01开始的字符串吗”，你必须交换所有接纳状态和非接纳状态。为了实现这一点，你必须使用图3–4中的自动机，而非图3–5中的。

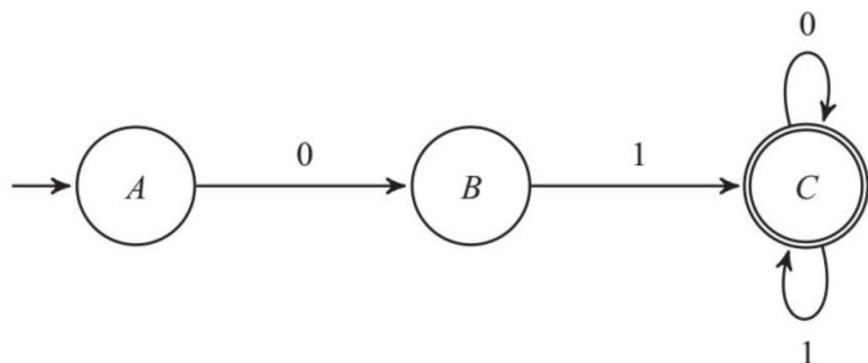


图3–5 剔除了陷阱的自动机

一些基本事实

什么是最简有限自动机？是不是有限自动机回答问题的方法不止一种？本节我们将简要探讨这两个基本问题。

最简有限自动机

最简有限自动机是只有一个状态的有限自动机。如图3–6和图3–7所示，机器M_5接纳所有由0和1组成的字符串，而M_6拒绝所有字符串（M_6的图中只有一个不是接纳状态而是陷阱状态。我们的方法是忽略非接纳陷阱以简化构图，但是在这种情况下，忽略这个状态并不能产生一张有用的图）。

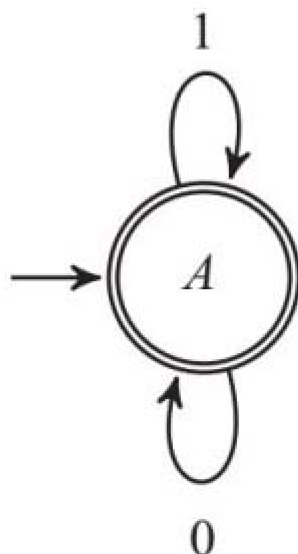


图3–6 接纳所有字符串的自动机

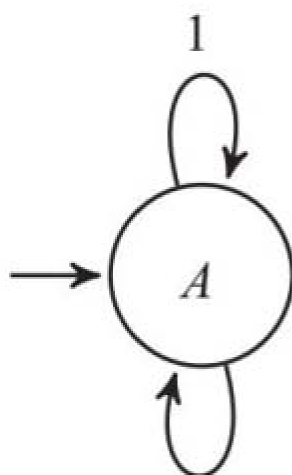


图3–7 不接纳任何字符串的自动机

等价自动机

人们很自然会想到这个问题：有限自动机回答问题的方法是否只有一种，或者是否可以设计两种不同的自动机来识别完全相同的字符串。一个简单的方法向我们展示了有可能存在两个等价但不同的有限自动机，即它们可以识别完全相同的字符串（如图3–8所示）。

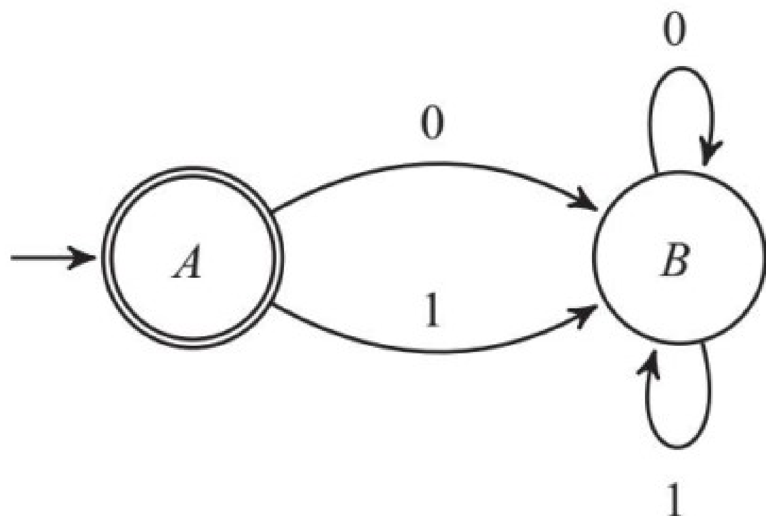


图3–8 只接纳ε的自动机

使用M_1来构建M_7，让所有状态变为接纳状态。因为M_7的每个状态都是接纳状态，所以它将接纳所有字符串。这意味着它与M_5是等价的。这告诉我们，设计机器回答一个问题的方法不止一种。当然，这也正是我们期盼的：回答一个问题的方法应该不止一种。

对有限自动机而言，我们可以证明存在唯一的含有最少状态数的机器（这句话可以理解为存在唯一的最简机器）。机器M_1有两个状态。我们不可能设计出少于两个状态的有限自动机来接纳具有奇数个1的字符串。因此，M_1是回答这个问题的含有最少状态数的机器，它是唯一的。如果你想要设计一个不同的机器来回答这个问题，它要么是M_1，要么有更多状态。

在结束这一节时，有一点需要注意。当我们判断两个机器是否计算等价的

时候，应该着重关注空字符串ε。尽管它是一个没有符号的字符串，但是它仍然是一个合法字符串。机器M_7帮我们明确了这一点。它接纳了ε，但是不接纳其他字符串。这个机器与“不接纳任何字符串”的M_7并不等价。

我们已经看到了一些有限自动机的例子，你可能会好奇这些机器究竟多强大。它们能解决什么问题？什么问题是它们解决不了的？我们将在接下来的两个小节中回答这两个问题。

正则表达式

斯蒂芬·克莱尼（Stephen Kleene）发现了一种巧妙的方法来描述有限自动机的语言——正则语言。

克莱尼是阿朗佐·邱奇的学生。他们一起提出了λ积分，这也是邱奇在其关于判定问题的著名论文中使用的方法。随后，克莱尼定义了正则表达式，并且证明这些表达式描述的就是有限自动机的语言。

正则表达式除了使用输入磁带的字母表，还会使用一些额外符号，包括：

$$(\quad)\quad +\quad *$$

括号用于将内容组合起来，其他两个符号有各自的意义，但加号“+”并不代表加法，乘号“*”也不代表乘法。

+用于表示左侧或右侧的可能的表达式。所以，表达式0 + 1代表两个字符串：0和1。表达式0（0 + 1）代表两个字符串：它们以0开始，第二个符号是0或1，即00和01。因此，我们可以写出0（0 + 1）= 00 + 01。（0 + 1）（0 + 1）代表四个长度为2的字符串，开始是0或1，结束是0或1。因此：

$$(0+1)(0+1) = 00+01+10+11$$

我们用ε来表示空字符串。表达式（ε + 0 + 1）（0 + 1）代表字符串以ε或0或1开始，以0或1结束（请注意，ε0表示的意思与0相同，即ε0 = 0。类似的，ε1 = 1）。

因此我们得到：

$$(\varepsilon+0+1)(0+1)=\varepsilon 0+\varepsilon 1+00+01+10+11$$
$$=0+1+00+01+10+11$$

符号*被称作星号或克莱尼星运算。这个运算表示将表达式与其自身进行有限次连接（其中也包括0次）。表达式0^*代表任意0的有限次数连接，即

$$0^* = \varepsilon + 0 + 00 + 000 + 0000 + \ldots$$

一个使用星运算的常用表达式是$(0+1)^*$，这等价于

$$\varepsilon + (0+1) + (0+1)(0+1) + (0+1)(0+1)(0+1) + \ldots$$

也等价于

$$\varepsilon + 0 + 1 + 00 + 01 + 10 + 11 + \ldots$$

这里列出了所有能由0和1组成的字符串。

举个例子，如果你想列出所有“以两个1开始，以三个0结束”的字符串，你可以写出$11(0+1)^*000$。

正则表达式描述了可能输入符号的字符串，它被定义为任何能够使用这些符号组成的表达式。

斯蒂芬·克莱尼证明，给定任意正则表达式，可以设计一个有限自动机来接纳由这个表达式定义的字符串。他还证明了，给定任意有限自动机，它能够接纳的语言可以用正则表达式来描述。从这点来看，我们可以认为正则表达式和有限自动机是等价的。它们都描述了完全相同的字符串集合。

M_2的语言是所有以01结束的字符串。我们可以用正则表达式$(0+1)^*01$来表示。M_4的语言是所有以01开头的字符串。它的正则表达式是$01(0+1)^*$。对正则语言而言，存在不止一个有限自动机和不止一个正则表达式来描述同一种语言。

一些简单机器可能拥有相当复杂的正则表达式。M_1的语言就是一个例子。这个语言包含由奇数个1组成的所有字符串。如何用正则表达式来描述它，不是那么容易就能想清楚的，但是我们将展示如何一步接一步地将这个机器转化为一个正则表达式。克莱尼对正则表达式和有限自动机的证明分两部分。在第二部分，他证明了相反的过程。在我们的例子中，将M_1转

变为正则表达式，涉及克莱尼第一部分证明中的关键步骤”。

这个机器从状态A开始。如果字符串被接纳，它必须在状态B停止。我们现在认为，当它首先进入B时可能已经读入了符号。或许它已经绕着0的圈子转了很多圈，但是为了来到B，它必须读入一个1，这可以写作0^*1。机器现在处于状态B。它或许会在某一点离开状态B，之后会返回。这导致它可能读入全部由0组成的字符串停留在B，并且在读入一个1后，又回到状态A。这些字符串也可以是0^*1。和前面一样，从A返回B的字符串可以由0^*1表示。这意味着，如果机器离开状态B又返回，它必须读入字符串0^*10^*1。当然，这个机器也有可能永远不离开状态B，或离开状态B很多次，可以用（$0^* 10^* 1$）* 来表示。机器最后一次从状态A变换到状态B，它可能会停留在状态B一段时间。这对应机器在读入由0组成的字符串。我们可以用0^* 来表示。把这些表达式组合起来，我们得到了描述M_1的语言的正则表达式：0^*1（0^*10^*1）$^*0^*$ 。

克莱尼的结论为我们揭示了有限自动机可以做什么。它们识别由正则表达式定义的字符串，这通常用于词语和文字处理。例如，我们有一篇文章，其中一部分是用美式英语写的，一部分是用英式英语写的，我们想要找到这些词，并用*red*取代*color*或*colour*，我们可以搜索字符串$colo$（$\varepsilon + u$）r。我们使用正则表达式输入搜索字符串，随后找到指定字符串，计算机就可以模拟对应的有限自动机。

正则表达式描述了模式。使用这些表达式的工具成为几乎所有现代编程语言的组成部分。知道如何使用这些表达式是一项重要的实践技能。

现在我们有两种使用有限自动机的方法，一种是设计一个有限自动机，另一种是设计描述相关语言的正则表达式。尽管从理论上说，任何关于有限自动机的问题都可以用这两种方法中的任意一种来回答。但是在实践中，某些情况下使用其中一种要比另一种容易。我们从M_1的语言中可以看到，使用自动机来表示相当简单，而使用正则表达式则相当复杂。再考虑两个问题：你能设计出一个有限自动机来接纳开始是000，结尾是111的字符串吗？正则语言的补集是正则语言吗？

对于第一个问题，我们使用正则表达式可以轻松解决，答案是000（$0 + 1$）* 111。这个表达式明确描述了这个语言。为这个语言设计一个有限自动机似乎相当困难。

对于第二个问题，我们已经在前面回答过：交换有限自动机的接纳状态和非接纳状态。在这里，使用有限自动机很容易就能得出答案。使用正则表达式来回答这个问题看起来则相当困难。

我们在探讨算法的时候会重新讨论这个观点。我们将获得很多等价的定义。每个定义都给了我们不同的观点，以及理解和思考计算真正含义的不同方法。一些从某个角度看似乎相当晦涩的问题，换个角度就会豁然开朗。

有限自动机的瓶颈

现在，让我们回到这个问题：什么问题对有限自动机而言太过困难？我们想要设计出一些问题，让我们有充分证据证明这些问题超出了有限自动机的能力范畴。只设计一些我们感觉“超出有限自动机能力范畴”的难题还不够，我们想要得到证明。为了设计这些证明，我们需要知道有限自动机的限制因素有哪些。

最主要的限制基于一个事实：存储新一位信息的唯一方式是进入一个新状态。有限自动机只有有限数量的状态，因为它们只能记住有限数量的字符串。一个有限自动机能区分的不同信息的最大数量，等于这个机器拥有的状态数。例如，机器M_1只需要两个状态，因为我们只需要两种信息——0的数量是偶数还是奇数。

给定一个特定有限自动机，我们知道它的状态数是确定的——这个数字可能非常大，但一定是确定的，让我们用s来表示这个数字。因此，这个自动机最多能够区分s种信息。无论s有多大，我们很容易就能设计一个需要最多$s+1$种信息才能解决的问题。下面，我们会给出两个例子。但是，我们先来介绍一下在接下来的证明中将会用到的鸽巢原理。㊟

鸽巢原理与将物体放入容器有关。这个原理用显而易见但有效的方法说明，如果物体数多于容器数，那么一定存在一个容器会容纳不止一个物体。

同样数量的0和1

问题是，给定一个由0和1组成的字符串，0和1的数量是否相同？

这个问题超出了有限自动机的能力范畴。为了搞清楚原因，我们首先考虑一个有5个状态的自动机能否回答这个问题。答案是不能，因为无论我们怎样设计有5个状态的自动机，我们总是能设计两个输入字符串——一个有等量的0和1，另一个有不等量的0和1，是5状态自动机无法区分的。

假设我们有一个5状态自动机。考虑一下6个输入字符串：0，00，000，0000，00000和000000。我们正在这个机器上运行6种输入。让我们来看

看这6种输入最后会使这个机器进入5种状态中的哪一种。鸽巢原理告诉我们，至少有两种输入最后会在同一个状态停止。假设将00和00000输入机器后，它们会在同一个状态停止。如果出现这种情况，那么新的字符串0011和0000011也会在同一个状态停止。我们现在遇到了麻烦。这个机器是否同时接受或拒绝0011和0000011，取决于这个最终状态是否是一个接纳状态。如果我们的机器能回答“给定一个字符串，1的数量是否等于0的数量”这个问题，它应该可以接纳0011，并且拒绝0000011，但是它不能。

尽管我用特定例子说明了这个论据，但是设计一个含有任意数量状态的任意自动机更直观：给定任意有限自动机，输入0、00、000……确定其中两个字符串最终在同一个状态停止（一旦字符串的数量超过了状态数，鸽巢原理就能确保这种情况一定会发生）。这两个字符串有不同的长度。给两个字符串添加同样数量的1，就可以让其中一个新字符串获得等量的0和1。很明显，另一个字符串的0和1的数量不相等。机器无法区分这两个字符串。

平衡括号

词语处理器通常拥有一个判断表达式中的括号是否匹配的内置功能。举个例子，（（（）（））（））是平衡的[注]，但（））（（）不是。为了使括号平衡，有同样数量的左右括号还不够，它们必须以正确方式配对，即一个左括号对应一个右括号。平衡括号问题是：给定一个由“（”和“）”组成的字符串，它们是正确配对的吗？

尽管这个问题与之前讨论的数字0和1的问题不同，我们还是可以借助与上文相同的思路，证明这个问题是有限自动机无法解决的。这里，我们不再输入由0组成的初始字符串，而是输入“（”字符串。余下的论证就很清晰了。

尽管这两个问题都超出了有限自动机的能力范畴，但是都有简单的算法可以解决。我们将在下一章中进行说明。读者可以自己思考一下如何一步接一步地设计解决这些问题的方法。给定一个由括号组成的字符串，如何判断它们是否平衡？[注]

磁带和配置

如果我们要设计一台现实中的有限自动机，它可以使用各种方式来读取输入。例如，它可以使用蓝牙或Wi-Fi（一种允许电子设备连接到一个区域

网的技术）建立连接，但是我们不用真的构建一个实体机器，因此在选择输入方法的时候并没有约束，只要这个输入方法易于可视化、使用方便即可。图灵介绍了纸片磁带输入，我们将使用这个方法。

考虑一下M_1上进行的计算，也就是我们在本章进行的第一个计算（如图3–9所示）。

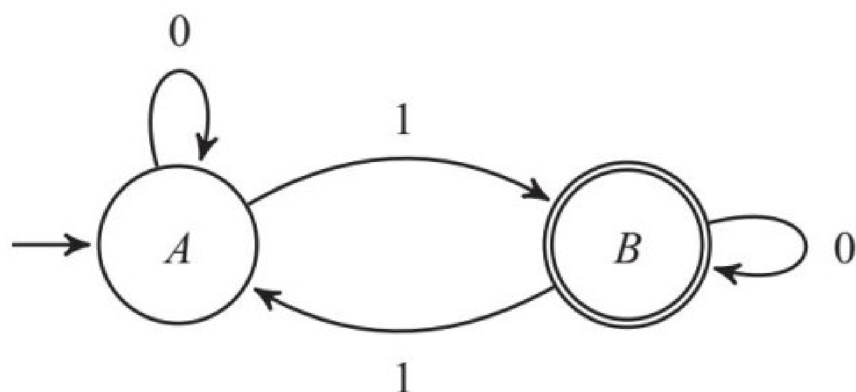


图3–9 有限自动机

像之前一样，我们输入100。假设这个输入是写在磁带上的，M_1有一个磁头，可以从磁带上读取符号。磁头沿着磁带移动，每次只向左或向右移动一个单元。我们用机器的当前状态来标记磁头。图3–10描述了机器停止前的三个步骤：

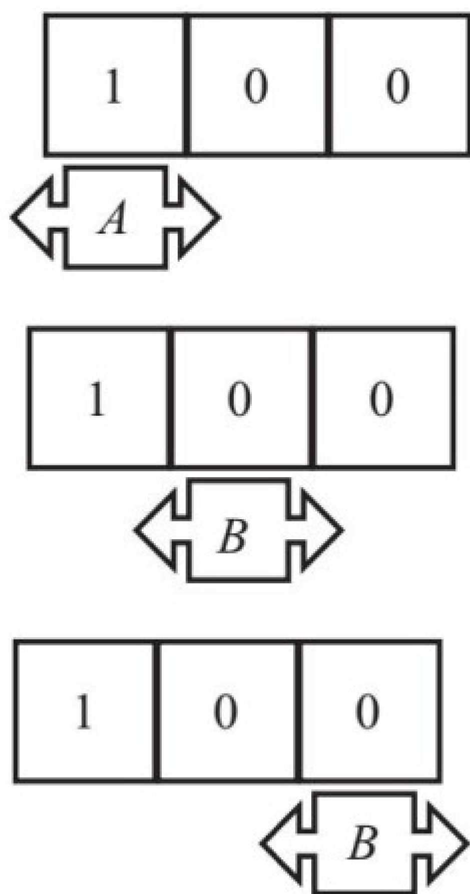


图3–10 磁头读取步骤

我们可以给出相同的信息，不必画出磁头和磁带。相比于直接在被读入的符号下方标记状态，我们可以在右侧列出机器状态。这个计算可以表示为$A100$，$1B00$，$10B0$，$100B$。每个字符串告诉我们有什么写在了磁带上，磁头处于什么位置（也就是下一步将读入哪个符号）以及机器的当前状态。这些字符串被称作配置。

我们会根据配置再次列出这个计算。这一次，一个配置写在另一个配置的上方。这种形式将帮助我们看到它们与对应问题的联系。

$$A100$$

$$1B00$$

$$10B0$$

$$100B$$

联系对应问题

我们会继续研究机器M_1，以此来解释这种方法与波斯特对应问题的联系。如果我们处于状态A，并且输入一个0，那么我们会停留在状态A。这意味着，如果我们在计算的某个节点获得的配置为$\ldots A0\ldots$，下一个配置会是$\ldots 0A\ldots$。更准确的说法是，$A0$后面一定是$0A$。$A1$后面是$1B$，$B0$后面是$0B$，$B1$后面是$1A$。这些信息可以用图3–11的方式给出：

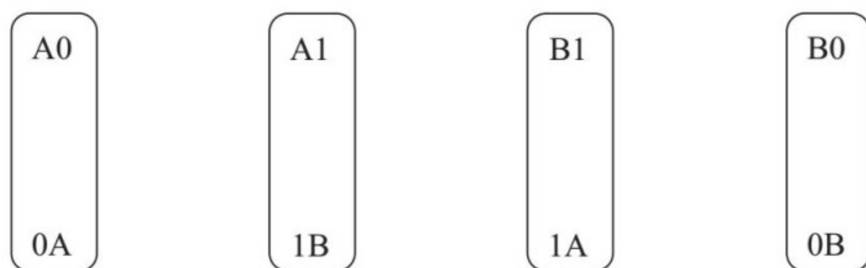


图3–11 对应问题

这其中有很多技巧。第一个技巧是设置一块瓷砖，在上方写*，在下方写*A。这迫使我们从这块瓷砖，以状态A开始。我们还需要为接纳状态设置一块瓷砖，在上方写B，下方是空白。完成之后的瓷砖如图3–12所示：

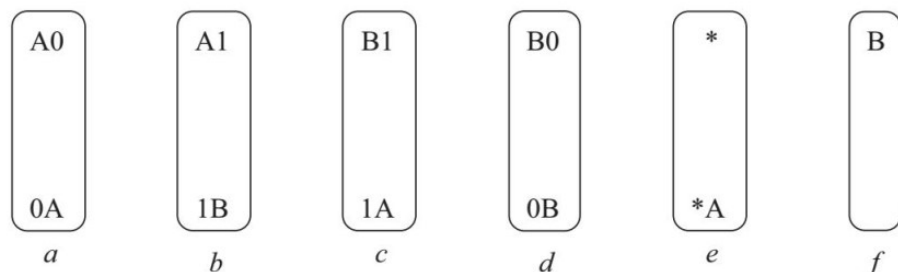


图3–12 瓷砖设置技巧

我们将找出一种解决对应问题的方法（如图3–13所示）。我们将这些瓷砖排成一行（可能会有重复），以便让顶部的符号序列等于底部的符号序列。

我们必须从瓷砖e开始，这是唯一一块含有初始符号的瓷砖。之后，可以使用瓷砖a或b。我们将发现与输入100相对应的对应问题的解决方案。因此，我们选择瓷砖b。对于接下来的瓷砖，我们有三种选择，分别是c、d和f。对应我们计算的瓷砖是d，请注意，f也可以解决对应问题（它对应字符串10）。接下来，我们还是有三个选择：c、d和f。我们还是选择d。和前面一样，我们可以选择c、d或f，但是我们这次选择f，得到答案。

第一块瓷砖要求我们必须从状态A开始。其他瓷砖分别对应上一节结尾提到的配置。

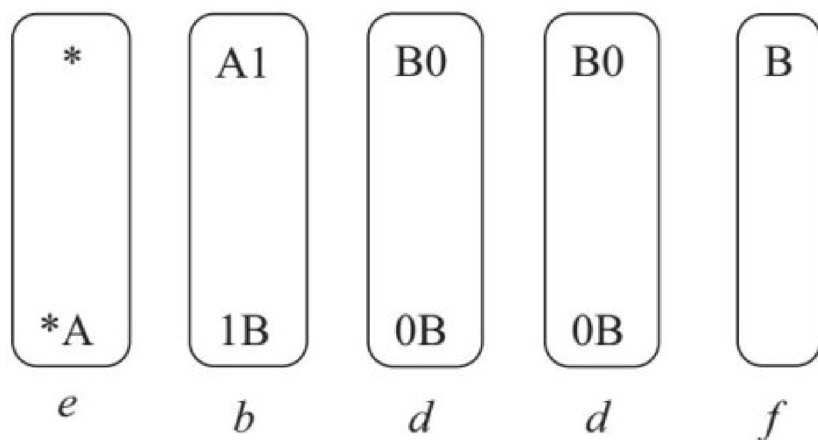


图3–13 对应问题的解决方案

给定任意机器M_1接纳的由0和1组成的字符串，存在一个使用瓷砖集合方法给出的对应问题的等价解决方案。反过来，任意对应问题的解决方案都可以给出令M_1接纳的0和1的字符串。

有限自动机是非常简单的机器，也很容易理解。这些机器不会涉及不可判定的问题。我们可以找出算法来判断两个自动机是否等价，因此这些瓷砖的对应问题都是可以确定的。

尽管这些机器的能力有限，但只需稍加改动，就可以让它们成为强大的计算机器。这是我们将在下一章讨论的内容，我们将允许机器在磁带上书写符号。这个微小的改变将会给计算能力带来巨大的变化。

---

1. 即配对的。——译者注

2. 1948年，维纳撰写了经典书籍《动物和机器中的控制论或控制和通信》。

3. 不久后，冯·诺依曼撰写了一篇内容广泛的论文《自动机的一般逻辑理论》，从这时起，计算机的理论模型开始被称作自动机。

4. 1963年，ASCII被发明出来，后来成为英语世界国家延用多年的标准编码方式。但是，这种方法只能使用7位编码，$2^7 = 128$个字符。不同国家有不同的字母和符号，无法使用ASCII编码这些内容。因此，必须修改编码方式。随着互联网和万维网的发展，以一致方法拓展编码变得越发重要——有必要在全世界范围内，保证每个编码字符串表示的内容一致。当前，新的标准是Unicode。这是一个将字符和符号对应数字的列表，它的前128个内容与ASCII一致。这个列表一直在持续更新。截至2015年6月，这个列表包含超过100万个实体，包含129种不同的编写脚本。

5. 奇偶校验检查有时被嵌入条形码。这种设计是为了正常读取条形码。奇偶校验位被用于判断正从哪个方向读取条形码，或者条形码正从左向右被读取，还是从右向左被读取。

6. 证明的第二部分包括将正则表达式转换为非确定性自动机，之后证明非确定性自动机与确定性自动机拥有完全相同的算力。这点并不困难，通常在标准的本科生课程中都有涉及，但因为太过冗长，本书不会包括这些内容。

7. 杰弗里·弗里德尔（Jeffrey Friedl）的《掌握正则表达式》就是证明正则表达式在各种实践应用中的有用之处的流行书籍之一。

8. 美国的“办公室邮箱”（office mailboxes）就是英国所谓的“鸽

巢”（pigeonholes）。

9. 我们只会研究两种自动机：有限自动机和图灵机。在计算理论的标准课程中，通常还要研究下推自动机（pushdown automata），一类算力比有限自动机强大，但差于图灵机的机器。关于等数量0和1的问题以及平衡括号的问题，都可以使用下推自动机来解决。超出下推自动机能力范畴的问题是使用由三个字母构成的字母表的字符串，并且询问字符串中每个字母的数量是否相等。

第四章 图灵机

> 或许我们可以这样解释隐藏在数字计算机背后的理念——这些机器旨在执行人类计算者能够完成的所有操作。注
>
> ——艾伦·图灵

现在，让我们回到1935年的剑桥，来到图灵身边。他提出了一个判定问题，这个问题超出了所有算法的能力范畴，使算法无法在所有情况下做出正确解答。图灵意在证明希尔伯特是错误的。因为没有定义能告诉我们一个程序作为算法到底意味着什么，图灵的第一步就是要明确地定义这个问题。

在当时，“computer”这个单词指的是从事计算工作的人。图灵的方法是观察人类计算者如何处理计算，然后设计一个理论机器来执行完全相同的工作。正如1936年图灵在论文中阐述的那样：

> 一般而言，计算是通过在纸上书写某些符号完成的。假设我们将这张纸划分为小学算数本那样的方格。在小学算术中，这张纸的二维特征有时会被使用。但是我们通常会避免这样的用法，而且我认为大家都会同意，纸的二维特征不是计算的实质。我们假设计算是在一维纸面（一个被划分为方格的磁带）上执行的。我们还要假设，可能被输出的符号的数量是有限的……计算者在任意时刻的行为，都是由他在那一时刻观察到的信息和心理状态的符号确定的……我们假设计算者执行的操作被划分为“简单操作”——这些操作十分基础，并且无法进一步细分。这些操作包括物理系统的一些变化，比如计算者和他的磁带的变化。如果知道磁带上的符号顺序，我们就可以知道系统的状态，这些符号的顺序可以通过对计算者和计算者的心理状态观察而知。我们可以假设，在一个简单的操作中，至多有一个符号被修改。

我们将借助图灵的方法，从分析人类如何进行计算开始。请考虑一下这个问题：

$$3457$$

$$+\ 237$$

在处理这样的问题时，我们首先会识别到加号“＋”以及格式，认识到这是一个加法问题。我们将此视作已知，并且仔细观察如何处理这个计算。我们知道要从最右侧的一列开始，但是如果打算细分所有细节，就需要解释我们是如何进行定位的。我们从左至右浏览不同的列，在看到一列空格时，最右侧的列是我们读入的最后一列。

我们通常不考虑空白符号，但是它们对计算十分重要。空白列告诉我们从哪里开始，到哪里结束。它们很重要，所以有必要使用一个符号显式地表示它们。我们将使用希腊字母β。这个问题可以被改写为：

$$\beta 3745\beta$$

$$+\ \beta\beta 237\beta$$

一旦我们定位了从哪里开始，就可以将那一列的数字加起来。存在两种可能的情况：一种是数字相加小于10，在这种情况下，需要进位给下一列的数值就是0；另一种是数字相加后等于或大于10，在这种情况下，进位给下一列的数值就是1。在两种情况下，我们都写下单位数，移到左边的下一列（在我们的例子中，5 ＋ 7得到进位数1，单位数2）。我们重复这一过程，将列中的数字与进位数相加；写下单位数，并且计算新的进位数。继续下去，我们可能会遇到一列含有一个空格符号和非空格符号（在我们的例子中，这一列包括3和空格）。在这些情况下，就相当于0 ＋ 3 ＝ 3。

在某一步中，我们从一列向左移动，直到我们来到完全是空格的那一列。如果有来自前一列的进位数，我们写下1并且停止。如果进位数是0，我们什么都不写，只是停止。

这个例子说明了我们在计算时需要完成的任务——我们需要读入数据；我

们需要写入；我们需要将计算划分为有限数量的简单组成。这些组成就是图灵所指的心理状态，人们将其简称为状态。

人类计算者必须将数据作为输入。一般而言，这些输入被写在纸上或纸的不同页上，但是正如图灵提到的，这些同样可以写在磁带上。没有使用二维数据或指令的必要性，也没有合理的理由让我们一次读入不止一个符号。因此，将输入放在磁带上，并且拥有一次可以读入一个符号的磁头，就足以设计出可以变为计算者的机器。

他指出，在计算过程中，计算机将处于多种心理状态。计算者下一步会做什么，取决于他的心理状态以及他从磁带上读到了什么。通过这种方式，图灵了解了人类如何进行计算，证明了所有操作都可以由他的机器完成。实际上，图灵将计算拆解为不同的部分，并证明图灵机可以完成各部分的操作。

当时，图灵正在设计一台理论机器，他打算使用这台机器证明这些关于计算的结果——他并没有设计现实机器。他希望这台机器足够简单，所以他选择了可以读写的单一磁带和单一磁头，而没有分别使用输入和输出磁带。

你只需要增加一些附加性质，就可以将有限自动机改造为图灵机。我们通过审视这些需要做出的改变来开始我们的研究。第一个重要的改变是图灵机与磁带交互的方式。除了向右移动，磁头也可以向左移动（可能有些奇怪——磁带是静止的，磁头在移动。但是这就是图灵描述的工作方式，而且这一方式已经被广泛接纳）。第二个重要的改变是磁头既可以在磁带上写下符号，也可以读取符号。在磁带上写下符号使机器可以存储之后计算中将要用到的中间结果，这也为机器提供了输出最终结果的方法。

我们在描绘图灵机的构造时，将分析磁头从一个状态向另一个状态移动的箭头上都做了什么。例如，假设我们正处于状态A，从磁带上读到了一个0。对于有限自动机而言，我们只需要知道将进入哪一个新状态。但是对于图灵机而言，我们不仅需要知道将进入哪个状态，还要知道在磁带上写下什么，以及向哪个方向移动磁头。在我们的例子中，假设从磁带读入0后，我们移动到状态B，用X改写0，然后将磁头向左移动一步。我们将在图4–1中体现这些信息。

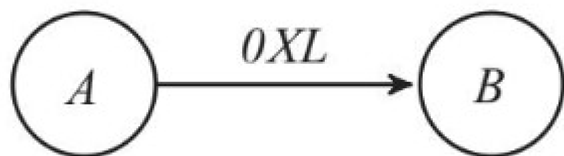


图4–1 图灵机示意图

在我们画出的图灵机示意图中，每一个箭头都会被标记一个三元符号：第一个是当前磁头正在读入的符号，第二个是磁头用来覆盖旧符号的新符号，第三个告诉我们磁头将会向左还是向右移动一步（L代表向左移动，R代表向右移动）。

图灵机的第三个重要的改变在于磁带的长度。人们经常将图灵机的磁带描述为无限长。这是因为我们不想由于磁带耗尽而导致计算结束。我们通常假设磁带的长度长到足以应对任何理论计算。举个例子，如果宇宙一共包含N个粒子，我们没办法拥有一个可以写上$N + 1$个符号的磁带。所以，我们无法执行一个需要写着$N + 1$个符号的磁带的运算。但是在计算理论中，我们只关注理论限制，而非物理限制。我们无法获得一个写着$N + 1$个符号的物理磁带，这个事实为我们揭示了一些有关宇宙的限制，但它没有告诉我们任何关于计算的理论限制。尽管我们说磁带是无限的，但在计算的任意一个节点，磁带上只会写着有限多个符号，磁带的其余部分是空白的。为了准确表示磁带上有什么，我们有必要使用一个符号表示磁带上的一个单元是空白的。例如，如果磁带上已经写了011，我们将表示为“…$\beta\beta\beta011\beta\beta\beta$…”，字符串两边的点表示无限空白。通常情况下，使用这些空白可以让机器知道它已经到达输入字符串的终点——与有限自动机在到达输入末尾时就会停止计算不同，图灵机可以继续运行。在这些例子中，我们需要某些符号来表示磁头来到了输入字符串的结尾，现在正在读入空白单元。

图灵机的第四个重要的改变是图灵机有唯一的接纳状态和唯一的拒绝状态。这些状态都是陷阱。一旦我们到达这些状态，就可以停止计算。如果在计算中到达接纳状态，我们会立即接纳这个字符串，即使这个字符串还有内容没有读入。如果我们到达拒绝状态，我们将立即拒绝这个字符串。

人们约定，当图灵机工作时，读写头从输入字符串最左侧的非空白符号开始读取。我将列举一些设计和运行图灵机的例子，以阐明这些见解。

图灵机的例子

我们从上一章第一个有限自动机的例子开始。一个机器可以识别由0和1组成的字符串中的奇数个1（图3–1对此进行了描述）。我们让图灵机拥有两个状态——状态A和状态B，A对应机器读入了偶数个1，B对应读入了奇数个1。对有限自动机而言，我们可以让B成为接纳状态。如果下一个输入是1，那么机器将移回状态A，表示有偶数个1输入。我们无法用图灵机完成这个过程。一旦图灵机进入输入状态，它将停止计算，接纳这个字符串。我们还可以让B成为“我们读入了奇数个1”这个状态，但是需要设置一个独

立状态，使其成为接纳状态。第三个状态，即接纳状态，我们用C来表示。现在，我们需要一种方式，让这个机器在读入一整个字符串后，接纳或拒绝这个字符串。一种方法是利用“输入字符串结束后，磁头将读入一个空白符”这个事实。当图灵机读入输入符号β时，我们将利用这点，引入一条从状态B到接纳状态C的箭头。

图4–2就是我们描绘的图灵机。在这个简单的例子中，磁头会在每一次操作后向右移动。所以，每个箭头上的第三个符号都是R。我们不需要在磁带上写任何东西，所以我们选择不处理这些符号。通过使用同样的符号进行复写可以完成这个过程。因此，每个箭头上标记的第二个符号与第一个符号相同。当我们读入这些符号的时候，就可以删除这些符号。在这种情况下，每个箭头上的中间符号如果都是β，在计算完成后，磁带会变为完全空白。

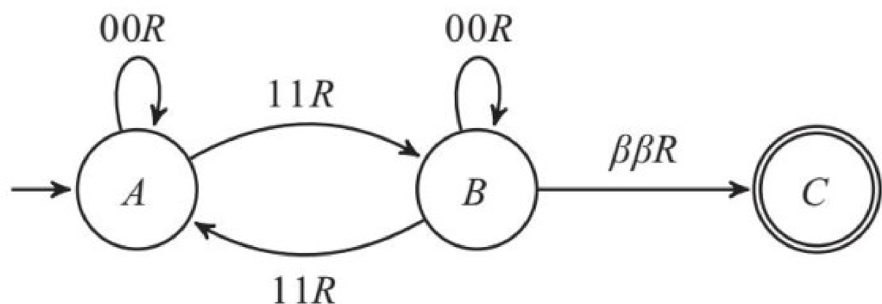


图4–2 图灵机模型

最终的结果是拒绝状态，相连的箭头会被忽略。如果我们想要引入停止状态，就会有一条标记“$\beta\beta R$”的箭头从状态A连接到停止状态。

算法

在下面的所有例子中，我们都将重复这个过程：描述一个算法来解决问题，再设计一个图灵机来实现这一算法。以我们刚刚讨论过的问题为例，算法应该是：从字符串的左边开始，每次向右移动一步。在每一步中，记录我们读入的偶数个1或奇数个1。一旦我们来到输入的结尾（即我们读入一个空白符），如果读入奇数个1，就接纳这个字符串，如果读入偶数个1，就拒绝这个字符串。

等量的0和1

我们想要一个算法来观察字符串中的0和1，判断它们的数量是否相等。有很多方法可以实现这点。你或许认为，我们能够记住0和1的数量，然后查看这两个数量是否相等。这对掌握技术能力的人类而言是可行的，但是对机器来说，这却不是最简单的处理方法。一个更加简便的方法是使用图灵机，将0和1配对，然后删除这对数字。不断重复这一过程，直到无法继续。如果我们删除了字符串中的所有符号，那么我们就可以知道，字符串中的0和1的数量相同，可以接纳这个字符串。如果字符串里有1或0剩余，我们就拒绝这个字符串。

这台机器将拥有写下新符号X的能力。它将使用这个符号来覆盖0和1组成的数对。

现在，我们来详细描述一下这台机器的操作：它将匹配0和1。如果首先读入1，它将用X取代1，并寻找对应的0。如果发现了0，它将用X取代0，并且返回输入字符串的起始位置。如果没有发现可以与1配对的0，它将在拒绝状态停止。如果机器首先读入0，它将用X取代0，并且寻找对应的1。

图4–3对这台机器进行了描述。理解这台机器上发生了什么，有助于理解每个状态的意义。

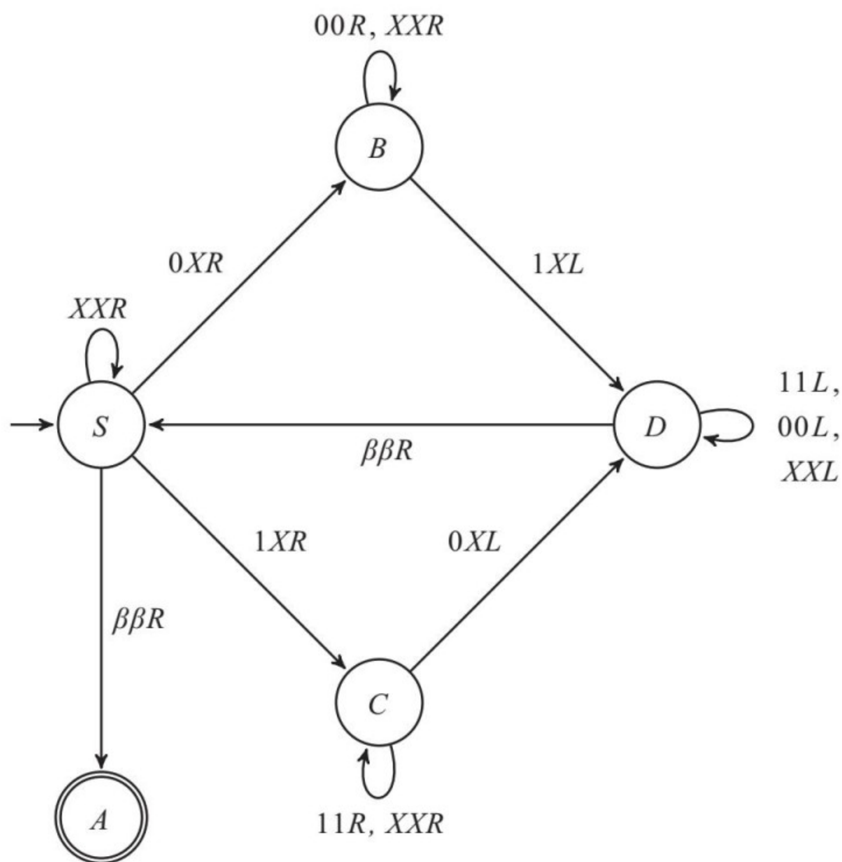


图4–3 判断0和1数量的图灵机

开始状态S说明：我需要将磁头不断向右移动，以寻找最左侧的0或1。如果发现了一个0，我将用X取代它，并且移动到状态B。如果发现了一个1，我将用X取代它，并移动到状态C。如果发现了一个X，我不会处理它，并停留在状态S。如果我没有发现0或1，并且读入了空白符，我知道它们一定都被配对了，所以我移动到状态A。

状态B说明：我刚刚读入了一个0，现在需要发现一个1来进行配对。我将磁头向磁带右侧移动。如果读入一个0或一个X，我会继续。如果读入一个1，我会用X取代它，并移动到状态D。

状态C与状态B类似，但是要把0和1互换。它说明：我刚刚读入一个1，现

在需要一个0来进行配对。

如果一个“0–1”数对被X代替，我就会进入状态D。这台机器需要为下一次计数进行重置，将磁头归位到左侧，重新进入开始状态。

状态D说明：我刚刚消除了一个由0和1组成的数对，必须回到开始状态。我继续将磁头向左移动，忽略其他符号，直到我读入空白符——这个符号说明我来到了左边的终点。我将磁头向右移动一步，获得第一个非空白符号，并且回到开始状态S。

在上一章中，我们没有描述拒绝状态，因为它是一个陷阱。如果你拥有一个0和1数量不等的字符串，那么你将在状态B或C里停止，并且获得一个β作为输入。这意味着这台机器达到拒绝状态，计算停止。

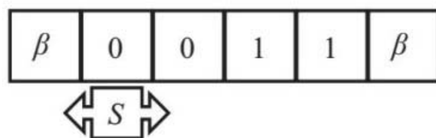
为了理解这台机器，我们设计了一些由0和1组成的短字符串。字符串0011是一个好例子。机器将在字符串两边加上空白单元。所以，我们的初始字符串是$\beta0011\beta$。最初几个状态如图4–4所示。

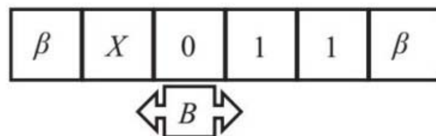
在这一阶段，这台机器已经完成了消除“0–1”数对的一次迭代，并且返回开始状态，将磁头归位到字符串最左侧。经过其他几次迭代后，剩余的0和1将被消除，来到图4–5状态。

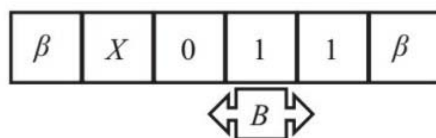
在这一点，磁头继续向右移动，机器停留在状态S，直到它到达右侧的空白单元。在此之后，这台机器将进入接纳状态，计算停止。

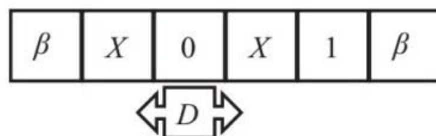
平衡括号

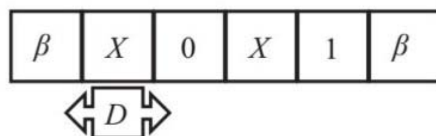
下一个图灵机关注平衡括号的问题。正如我们在上一章结尾提到的，这是另一个无法用有限自动机解决的问题。我们将使用上一个例子中用到的方法进行处理，定位一个左括号和一个右括号，用X取代这个括号对。但是，配对括号要复杂一些。我们不能单纯计算左括号和右括号的数量，查看两个数量是否相等，必须检查所有括号是否被正确地配对。例如，“ ）（ ）（ ”有相同数量的左右括号，但是它们并不平衡。

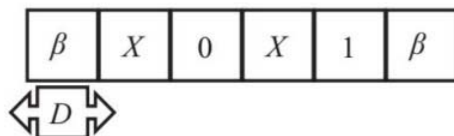

β
0
0
1
1
β
S


β
X
0
1
1
β
B


β
X
0
1
1
β
B


β
X
0
X
1
β
D


β
X
0
X
1
β
D


β
X
0
X
1
β
D

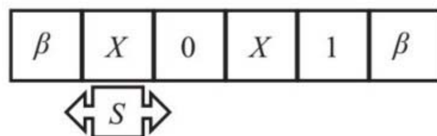

β
X
0
X
1
β
S

图4–4 短字符串初始状态

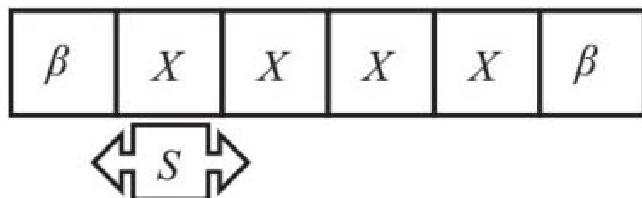


图4–5 短字符串经过迭代后的状态

确定这一序列是否平衡的一个方法，是向字符串右侧移动，直到你发现第一个右括号。用X取代这个右括号，然后向左移动。你读入的第一个左括号与你刚刚用X取代的括号配对，所以你把这个左括号用X替代。

图4–6描述了相关的图灵机。每个状态的意义如下。

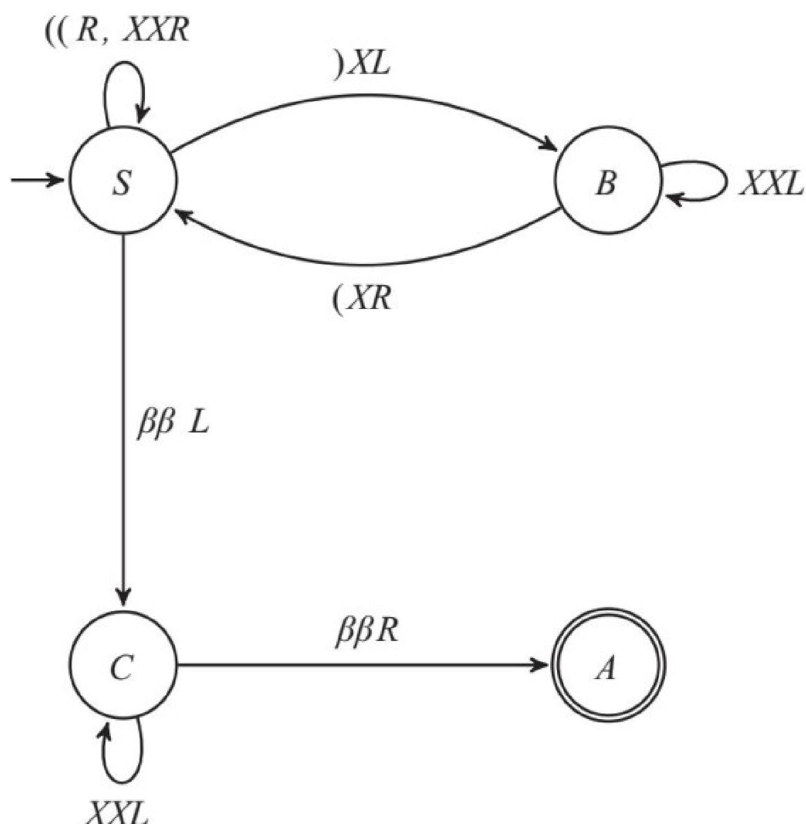


图4–6 平衡括号图灵机

状态*S*说明：我将继续把磁头向右移动，直到我读入右括号。当我读入一个右括号后，我将用*X*取代它，并且移动到状态*B*，如果我来到了字符串尾端，而且没有读入右括号，就知道没有剩余的右括号。我移动到状态*C*。

状态*C*说明：磁头在字符串的右端，没有剩余的右括号。我需要把磁头向左移动，检查可能剩余的左括号。如果我发现了一个左括号，我就会进入拒绝状态。如果没有发现左括号，我最终会来到空白单元，这告诉我们整个字符串都是由*X*组成的，于是我会进入接纳状态。

真正理解这台机器如何工作，有助于我们在其他情况下使用它。图4–7是我们在这个字符串上运行图灵机的最初几步（完整过程在注释中注）。



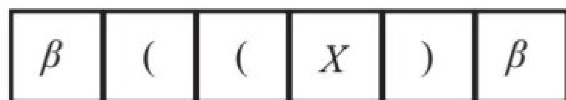














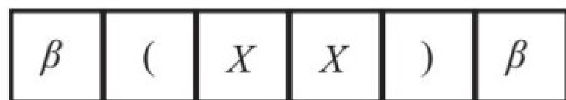




图4–7 在字符串上运行图灵机

可计算函数和计算

迄今为止，我们讨论的图灵机都被用于解决是非问题。当且仅当图灵机到达接纳状态时，它才接纳字符串。但是，图灵机拥有在磁带上书写字符的能力，所以它有输出答案的其他方式。它可以将答案留在磁带上。我们将简要地探讨这一理念。

可计算函数

自然数都是正整数。我们将使用一元符号表示这些数字。每个数值通过1重复出现的准确次数来表示（例如，4被写作1111）。我们观察那些图灵机，它们以有限个1组成的字符串为输入，在接纳状态结束，接纳有限个1组成的字符串。例如，我们可能有这样一个图灵机：初始字符串是1111，进入接纳状态后，在磁带上的字符串是111111。我们可以这样理解：输入数字4，输出数字6。我们可以将这些机器视作一种用于自然数的函数：输入一个自然数，输出一个自然数。这些函数被称作可计算函数。

这些函数在计算理论中起到了关键作用。我们的定义与图灵机有关，但是其他数学家和逻辑学家借助非常不同的方式设计了这些函数。在下一章里，我们将探讨邱奇的方法。这些不同的方法产生了相同函数的事实说明，它们是等价的。它们是同一事物的不同表达方式。

现在，你可能想看一些“将自然数转变为自然数”的可计算函数的例子，以及一些不可计算函数的例子。你能想到的任何例子都是可计算的。例如，包含加法、乘法、幂运算和指数运算的函数都是可计算的。事实上，不可计算函数很难被描述，但是当我们介绍忙碌的海狸函数时，就可以看到一个不可计算函数的例子（我们将定义这个函数，并且在分析不可判定的问题时，证明它为什么是不可计算的）。

计算

我们同样设计了可以进行常规数学运算的图灵机。设计一个图灵机来完成两个自然数的加法并不困难。如果我们想要计算$2+3$，以一元形式书写就是$11+111$。机器将会输出11111。设计这个加法机器的一个简单方法是消除第一个1（用β复写这个1），之后用1复写“$+$”（加号）。

我们还可以设计一个乘法机器。在这个例子中，如果我们输入1111×111，输出111111111111。下面，我们将简单地描述这个机器如何进行运算。

第一步是消除最左侧的1，然后将磁头向右移动到“×”（乘号），并且用X符号取代所有1，我们将获得$\beta 111 \times XXX$。

第二步是将磁带左侧的第一个1消除，将磁头移动到“×”的右侧。每个X都被换为Y，但是每进行一次从X到Y的转换，我们就在字符串结尾增加一个1，我们将获得$\beta\beta 11 \times YYY111$。

我们再次消除最左侧的1；将Y变回为X，并在字符串结尾增加1，结果是$\beta\beta 1 \times XXX111111$。

再重复一次，我们会得到$\beta\beta\beta \times YYY111111111$。

这个机器现在开始寻找最左侧的非空白符号。它读入了“×”。这让它将所有Y和X换为1，并且停止。我们最后得到的结果是111111111111。

减法和整数除法的编码并不困难。一旦拥有了可以计算加法、乘法、减法和除法的图灵机，我们就可以将它们结合为一个能够进行四则运算的图灵机。

邱奇—图灵论题

正如我们在本章开始提到的，解决判定问题的第一步是给出一个算法的定义。图灵考虑了计算应该包含的内容，提取了基本步骤，并且使用它们来定义图灵机。之后，他将算法定义为任何能够由图灵机计算的事物。

这是一个定义，因此它无法被证明或推翻，但你能够判断这个定义是否准确。人们通常觉得算法是进行计算的步骤。图灵给出了这个步骤的正式定义。如果你认为这个定义是正确的，就必须接受所有能够由图灵机实现的算法。如果你认为这个定义是错误的，就应该想出一个更好的定义，并且给出一个例子，让人们认为这才是算法，但是图灵机无法实现。

两种方法都有人尝试过，最终人们达成了一致，认为图灵已经抓住了一个程序作为算法的本质意义。当然，还存在算法的其他定义。回忆一下邱奇在图灵之前发表的关于判定问题的论文。邱奇同样需要定义什么是有效程序。他通过λ积分做出了定义。图灵在阅读邱奇的论文后意识到，共有两种不同的定义。研究之后，他发现尽管两种定义看起来非常不同，但它们却是等价的。实际上，它们定义了完全相同的概念。他快速修改了自己的论文，赶在发表前增加了对这种等价性的证明。

邱奇—图灵论题是这两种定义都抓住了“一个过程作为算法的意义”的本质。通常，这个定义是这样表述的：任何能够由图灵机计算的事物。

除了λ积分，还有其他关于有效程序的定义。波斯特定义了正则系统，克莱尼定义了一般递归系统。这些例子都被证明与邱奇和图灵的定义等价。这种等价性意味着尽管使用的方法不同，它们描述了完全相同的功能或性质。

在数学中，用于描述概念的标记非常重要。例如，牛顿和莱布尼茨同时发现了微积分，但是他们使用的标记却非常不同。事实证明，莱布尼茨使用的标记远比牛顿用于复杂计算的标记高级。注莱布尼茨使用的标记，一种名为u替代法的整合技术非常容易学习。u替代法实际上应用了链式法则，但是这个标记的高明之处在于隐藏了这个事实。

正如勃兰特·罗素所说："一个好的标记同时具备精妙性与建议性，有时这会让它看起来像一个栩栩如生的老师。"注这些评论与图灵和图灵对算法的定义极为契合。许多人以截然不同的方式定义和描述了算法，但是图灵的描述非常容易可视化，这为计算研究提供了一个简单的框架。在邱奇和图灵撰写论文之前，哥德尔不能确定人们能否准确定义算法。甚至在读过邱奇的定义以后，他还是不能确定，但是看罢图灵的论文，他立即被图灵的方法说服。

计算能力

我们已经看到，图灵机是具备一些额外功能的有限自动机。有限自动机的计算能力相当有限——平衡括号超出了它们的能力范畴，而图灵机则可以执行任何算法。如果我们为图灵机增加更多功能，会让它变得更强大吗？例如，为图灵机配备两个磁带和磁头。我们可以预留一个磁头用于输入和输出。当我们使用它的时候，最好有一个键盘用于输入，一个显示器用于输出。但是，无论我们增加何种额外功能，机器的计算能力都不会得到提升。新机器或许更易于使用，但是它能计算的任何事物都可以由标准图灵机实现，因为图灵机是一个理论模型，没有对输入规模或计算所需时间进行限制，它能做的事情远超过任何一个受限于输入规模和寿命的人类计算者。

这些标记同样适用于量子计算机。注这些计算机通常被描述为计算领域的下一次飞跃——它们更加强大，有能力解决超出传统计算机能力范畴的问题。量子计算带来的进步是速度。如果现实的量子计算机被制造出来，它们或许能够以比我们目前的计算机快得多的速度解决某些问题。一些在传统计算机上需要指数时间才能完成的算法，在一台量子计算机上只需要多项式时间。我们将简要解释这种进步具备何种意义。

多项式时间

这里的时间指的是计算的步骤数。我们可以设想每一步需要一秒来执行。一般来说，步骤数的多少取决于输入字符串n的大小。对每个正整数n，将有许多长度为n的输入字符串。图灵机在处理某些字符串时所需的步骤数，可能多于处理其他字符串所需的步骤数，但是我们可以知道每个n对应的最大步骤数。这使我们获得一个关于n的函数$f(n)$，即完成计算所需的最大步骤数。之后，我们再去考虑是否存在一个多项式$p(n)$，对所有正整数n，都能使$p(n) \geq \mathrm{H}f(n)$。如果存在这样一个有边界的多项式，我们就可以说这个算法是一个多项式时间算法。

回想一下我们处理平衡括号时用到的算法。假设我们有一个长度为n的输入字符串。要判断这个字符串是否是平衡的，图灵机所需的最大步骤数是多少？我们需要在字符串两边使用空白单元，磁头不断在磁带上向左、向右移动，永不停止。一旦磁头开始向右移动，它就会一直向右，直到用X取代一个括号，或者计算结束。一旦磁头开始向左移动，它就会一直向左，直到用X取代一个括号，或者计算结束。向左和向右移动的步数之和一定小于等于$n + 1$。当且仅当字符串是平衡的，并且n个括号都被X取代时，这个值等于$n + 1$（多出的一个步骤是图灵机在检查整个字符串是否全部由X组成）。随着磁头向左、向右移动，它读入了磁带上的非空白符号，或许也读入了字符串左边或右边的一个空白单元。之后，每一阶段向左或向右移动的步骤数最多是$n + 2$。所以，这个计算中的总步骤数一定小于或等于$(n + 1)(n + 2) = n^2 + 3n + 2$。

因为判断长度为n的输入字符串是否平衡的算法所需时间上限为$p(n) = n^2 + 3n + 2$，我们已经证明，这个算法是多项式时间算法。

我们现在可以为判定问题分类。我们用P来表示能够通过图灵机在多项式时间内解决的判定问题，并且我们可以证明问题“这个字符串是平衡字符串”属于P。

非确定性图灵机

非确定性图灵机是图灵机的变型，它在某一时刻可能处于不止一个状态，所以它可以同时进行多个计算。我们在第一次介绍图灵机时，使用了图4–8来说明：当图灵机处于状态A的时候，从输入磁带上读入0，图灵机移动到状态B，用X取代输入字符，随后将磁头向左移动。

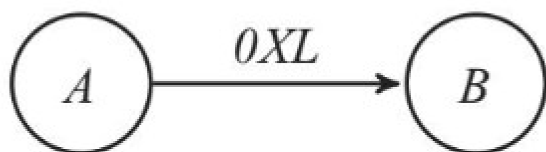


图4–8 图灵机

对非确定性图灵机来说，当它读入一个输入符号时，能够做的事有不止一个选项（如图4–9所示）。

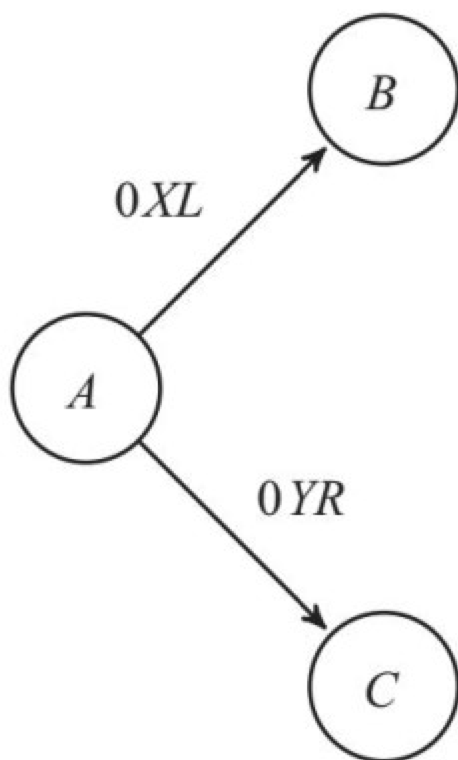


图4–9 非确定性图灵机

这意味着当机器处于状态A时，从输入磁带上读入0可以有两个选择：它可以移动到状态B，用X取代输入符号，并且向左移动磁头；也可以移动到状态C，用Y取代输入符号，并且向右移动磁头。我们必须记录计算的两种选

择（或分支）。每个阶段都可以有许多选择，所以计算通常会有许多分支。如果某个分支到达接纳状态，图灵机将会停止，输入字符串会被接纳。

需要强调的是，尽管名字叫作非确定性图灵机，它却是完全确定的。当我们描述这些图灵机时，通常会使用“选择”或“猜测”这类词。这些对初学者来说都相当有误导性。无论图灵机是否接纳输入字符串，它都是完全确定的。当然，想要证明输入字符串可以被接纳，我们只需要找到一条能够通向接纳状态的分支。在描述某个特定分支时，你必须在每一步做出正确选择，这就是我们常说的机器在“选择”或“猜测”正确选项。

非确定性图灵机会并行计算各个分支。因此，它通常比图灵机运行速度快，但是问题在于，它究竟有多快？是否存在这样一个问题，能够用非确定性图灵机在多项式时间内解决，但是无法用普通图灵机在多项式时间内解决？

我们用*NP*来表示非确定性图灵机可以在多项式时间内解决的判定问题。很明显，*P*是*NP*的子集，但是否存在某些属于*NP*但不属于*P*的问题仍然悬而未决。克雷数学研究所（Clay Mathematics Institute）悬赏100万美元征集这个问题的答案。一些人推测，量子计算机能够在多项式时间内解决的问题的集合，包括所有*P*以及一些不在*P*中但并非全部都是*NP*的问题。然而，这只是一个推测。我们必须证明*P*不等价于*NP*。

基于输入规模、处理计算所需的时间和存储量的理论是复杂度理论。这些问题不只是理论问题，而且具有重要的实际应用。大多数互联网公司需要加密信息的安全方法。然而，我们推测出但未证明过破解目前的许多加密方法要进行指数级运算。如果能够证明我们的互联网交易是绝对安全的，也是一桩好事。

量子计算机和非确定性图灵机或许在某些情况下速度更快，但是从计算能力来看，它们并没有比图灵机更强大。在本书中，我们将探讨计算机可以解决哪些问题，不可以解决哪些问题。要想回答这个问题，关注基本图灵机就够了。

不会停机的机器

迄今为止，我们讨论的范例图灵机都会停止。给定任何字符串，图灵机或者进入接纳状态，接纳这个字符串，或者进入拒绝状态，拒绝这个字符串。但是，依然存在其他可能性。我们有可能设计出一个永远不会停止的图灵机。

考虑一下图4–10中描述的TM_4。如果输入字符串以0开始，机器会移动到接纳状态，计算会停止。如果输入10，机器首先读入1，移动到状态B。随后，机器读入0，计算停止，进入拒绝状态。但是，如果我们输入11，机器首先会进入状态B，随后读入第二个1。接着移动到状态A，磁头向左移动，我们又回到了起始位置。这个过程会一直重复。计算永远不会结束。

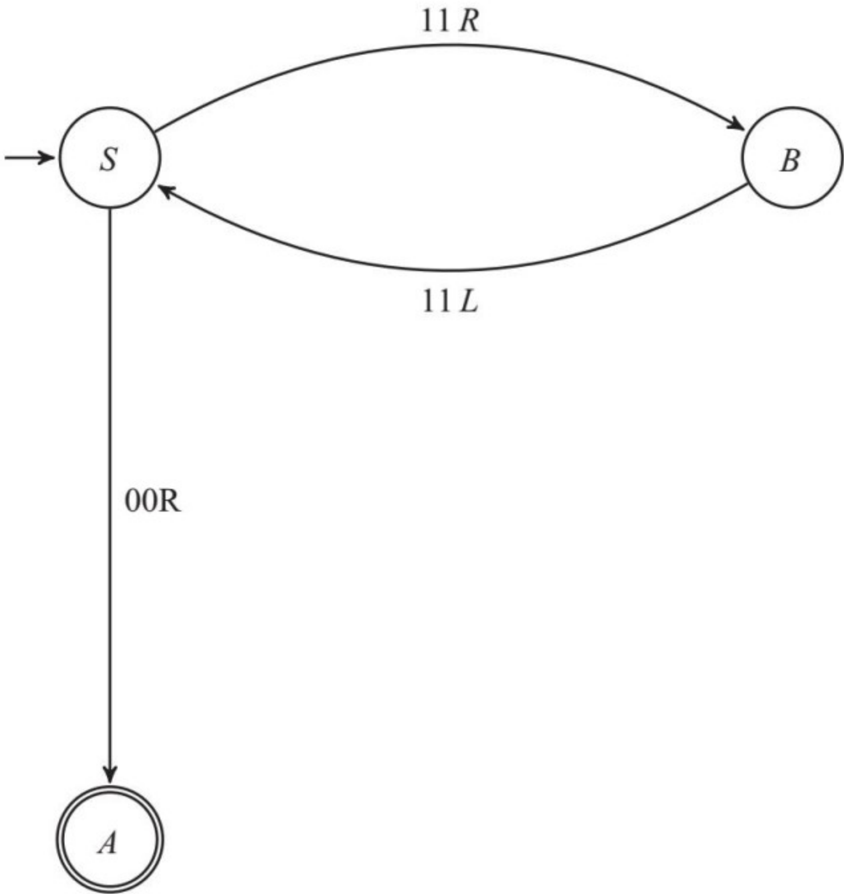


图4–10 永不停止的图灵机

因此，图灵机的计算有三种可能性：一个计算可以在接纳状态停止，也可以在拒绝状态停止，或者永远不会停止。如果一个计算永远不会停止，我们称这个图灵机发生了偏离。

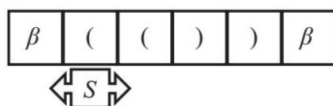
请注意，这一点和有限自动机有明显的区别。有限自动机获得一个有限输入字符串，字符串从左至右被读入，当来到字符串终点时，机器停止。因此，有限自动机总会停止。计算的步骤数等于输入字符串的长度。

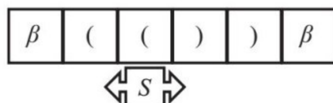
目前，我们尚不清楚偏离计算是否是必要的。某些人认为不会停止的机器设计得很糟糕，如果设计得更好，它们应该能够在所有输入上停止。但是，这不是问题的关键。事实是一些不会停止的计算并没有设计缺陷，这不是我们能够排除的。我们将看到，这种“某些机器会发生偏离”的性质对于我们能够计算和无法计算的事物有重要影响。

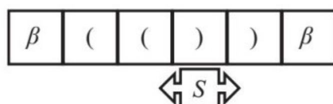
在继续讨论图灵机之前，我们会走一个小弯路，介绍许多人为算法和计算做出的非常不同的定义。虽然图灵的定义非常简洁，但其他等价定义为我们提供了审视计算的不同观点。正如之前提到的，一些从某个角度看很困难的事情，从另一个角度看就会变得简单。我们将在下一章讨论这些观点。

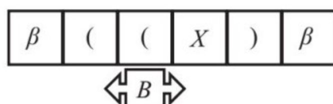
---

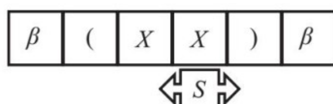
1. 艾伦·图灵，《计算机器和智能》。
2. 当图灵机TM_2在字符串(( )) 上运行时，图注–1是完整的运算过程。

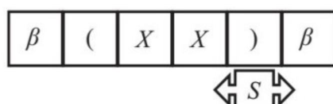

β (( )) β
S

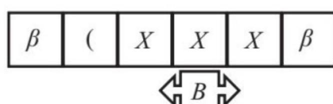

β (( )) β
S

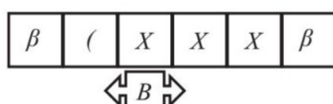

β (( )) β
S

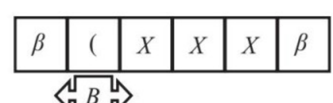

β ((X) β
B

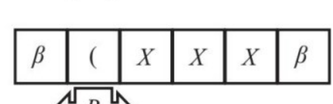

β (X X) β
S

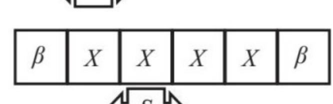

β (X X) β
S

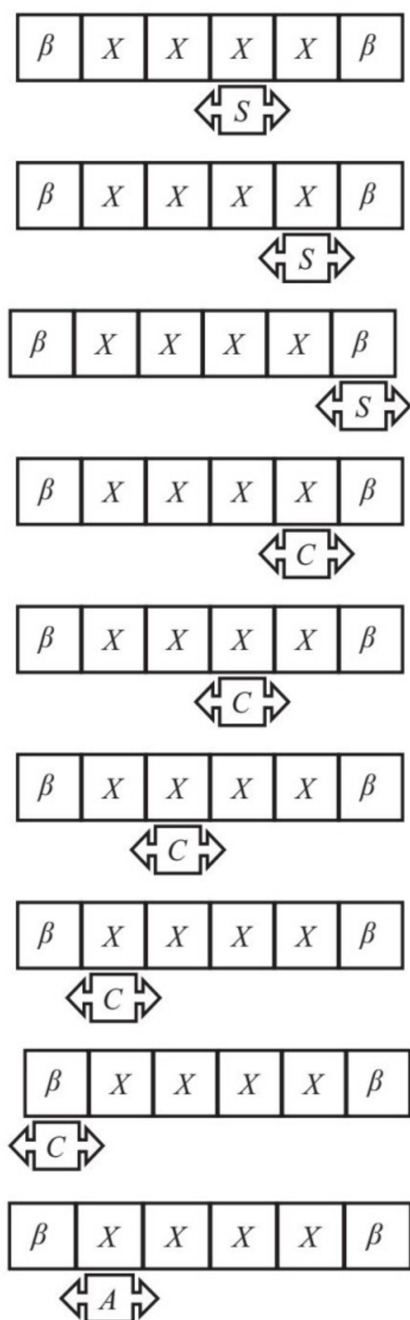

β (X X X β
B


β (X X X β
B


β (X X X β
B


β (X X X β
B


β X X X X β
S



图注–1 图灵机的运算过程

3. 莱布尼茨的标记是标准的 $\frac{dx}{dt}$ ，即x对t求导。在牛顿标记中，这被表示为$\dot{x}$。

4. 《维特根斯坦的逻辑哲学导论》。

5. 这并非微不足道的事实。1985年，大卫·多伊奇，这一领域的一位先驱，发表了一篇有关量子计算的里程碑式的论文《量子理论，邱奇—图灵原理和一般量子计算机》。在这篇论文中，他描述了量子计算机，并且介绍了它们与传统计算机的区别。问题在于量子计算机的算力是否比传统计算机强大。虽然会花上一些时间找出答案，但最终结论还是量子计算机的算力并没有更强大。

第五章 其他计算系统

> 我们可以这样说，分析引擎造就了代数模式，正如提花织机造就了花朵和叶子。
>
> ——阿达·洛芙莱斯（Ada Lovelace）

1823年，在英国政府充足的资金支持下，查尔斯·巴贝奇（Charles Babbage）开始制造自己的第一台差异引擎。当时的数学表格包含许多人类计算者导致的错误。巴贝奇的差异引擎是一台机械计算器，将提供一种更快捷、更廉价也更精确地生成这些表格的方法。巴贝奇设定了制造计划，但只雇用了一位工程师约瑟夫·克莱门特（Joseph Clement）来实际执行。这台机器结构复杂，这导致工程进展缓慢。随着时间的推移，巴贝奇和克莱门特的关系慢慢变得紧张。这种紧张终于在1831年爆发，他们意识到无法继续合作，项目随即停止。

尽管巴贝奇没有制造出实实在在的机器，但他设计出了一台新的差异引擎——差异引擎2和分析引擎。注分析引擎远比差异引擎复杂。它最重要、最具创新性的功能是可以通过穿孔卡编程。这是他从机械织机上借鉴来的想法。1801年，约瑟夫·玛丽·雅卡尔（Joseph-Marie Jacquard）设计了机械织机，这台机器可以根据一系列穿孔卡控制的操作，编织出复杂的样式。巴贝奇意识到，他可以在自己的机器上使用同样的理念。于是，他设计出了可编程计算机。

洛芙莱斯伯爵夫人奥古斯塔·阿达·金（Augusta Ada King），也就是现在大家熟知的阿达·洛芙莱斯，是诗人拜伦伯爵的女儿。她的母亲对拜伦一家的精神状态深感担忧，决定让自己的女儿学习数学，这使她得到了当时世界上最顶尖的数学家的教育和指导。因此，她和巴贝奇注定有一场邂逅。

洛芙莱斯对巴贝奇的机器很好奇。她不仅理解巴贝奇要做什么，而且看到了计算机的巨大潜力。她不是唯一一个受到巴贝奇启发的人。在意大利，数学家路易吉·梅纳布雷亚（Luigi Menabrea）同样对巴贝奇的分析引擎充满好奇。他根据自己在巴贝奇演讲上记录的笔记撰写了一篇文章。洛芙莱斯将这篇文章翻译为英文，并增加了自己的见解。她为梅纳布雷亚的文章增加的注释内容是原文长度的两倍。注在这些笔记中，她勾勒出自己对未来计算的看法。

这些注释之所以闻名世界，既因为其中包括第一个发表的计算机程序，也因为洛芙莱斯预见到计算机将获得数学以外的广泛应用，她是第一个站出来反对人工智能的学者。当时最大的计算机程序由巴贝奇和洛芙莱斯合作推出。人们对究竟是谁贡献了实质想法，谁应该得到赞誉各执一词。然而，洛芙莱斯通常被视作第一位计算机程序员。

她同样意识到，计算机可以在众多领域大有作为。她想到的一个应用领域是音乐。洛芙莱斯推测，计算机可以按照这些规则编程，并且能够“以任意复杂度编写精妙的乐章”。

她指出：“分析引擎并不会孕育什么东西。只要我们知道如何下达命令，它就能做到我们知道的任何事情。”这很明显就是在叙述现在我们所谓的人工智能。许多人，包括图灵，都认为她是错的，但令人惊讶的是，她在第一台计算机出现的一个世纪前就开始考虑计算机和大脑的关系。正如戴维·多伊奇（David Deutsch）指出的[注]，洛芙莱斯的错误并不在于她低估了计算机，而在于高估了人类大脑。

在继续对图灵机的研究以前，我们将探讨其他对字符串进行处理的方法：邱奇的λ积分、波斯特的标签系统以及元胞自动机。这几个方法都被证明可以用来进行计算。事实上，令人更惊奇的是，这些方法的计算能力都与图灵机等价。

标签系统能够进行任何计算，它的简洁性在证明计算系统的等价性时很有用处。例如，在证明元胞自动机可以完成任意一项图灵机能够完成的工作时，就会引入由标签系统模拟的图灵机。

我们还将研究一维元胞自动机。它生成了可以展现全部计算的二维图片。通过这些图片，我们有可能推测出隐藏规则的计算能力。

尽管本书围绕一篇写于1936年的论文展开，文中使用的标签系统和一维元胞自动机的例子都是近期才被提出的。撰写这种类型的图书很容易给人留下这样的印象：所有东西都已经做好了，没有什么需要创新的。图灵的研究揭示了完全相反的结论：在理论计算科学领域，相关研究永远没有尽头。因此，了解一些当代的结论同样大有裨益。

λ积分

邱奇提出λ积分的时候，哥德尔正在研究递归函数理论。克莱尼证明了这两个理论是等价的。邱奇认为，λ积分描述了所有可计算函数。因此，如果邱奇是正确的，哥德尔的递归函数也将能够描述所有可计算函数。哥德

尔很难相信，他和邱奇都在描述所有可以被计算的事物。随后，作为研究判定问题论文的一部分，图灵证明了他的图灵机理论与λ积分等价。所有人都认为图灵抓住了计算的实质。哥德尔意识到，邱奇实际上是正确的。

1937年，数学界出现了三种不同但等价的描述计算的方法。在随后的几年时间里，数学家接纳了递归函数理论，计算机科学家接纳了图灵的理论。邱奇的λ积分并没有得到太多关注。

邱奇是一位逻辑学家。他非常严谨，希望自己的学生能够以相同的方式研究逻辑。他是20世纪最重要的逻辑学家之一，他的很多博士生都在逻辑和计算机科学领域取得了出色的成绩。

λ积分使用自下而上的方法谨慎地构建了数学——所有东西都是由函数定义的。外行人发现，这种方法很难理解。甚至连克莱尼——帮助邱奇开发了λ积分的那个人，最终也不再使用这种方法。他的解释是，如果他用λ积分进行演讲，很难有人感兴趣，但是如果用递归函数描述完全相同的观点，他的演讲将会吸引很多人。然而，尽管最初并不受欢迎，λ积分在计算机科学领域还是起到了作用，并且仍然在起重要作用。

正如普林斯顿大学计算机科学尤金·希金斯讲席教授安德鲁·阿佩尔（Andrew Appel）所证明的，λ积分孕育了现代编程语言的许多概念。注1958年，Algol是根据λ积分理念开发的第一种编程语言，Lisp在开发过程中显式地使用了λ积分。大多数现代编程语言都是从这两种语言发展而来的。正如阿佩尔所说："变量绑定、范围、函数、参数传递、表达式和类型检查的概念都是直接从邱奇的λ积分中借鉴过来的。"

皮亚诺算术

你如何定义自然数？如何定义加法？这些是19世纪时人们提出的问题。1861年，赫尔曼·格拉斯曼（Hermann Grassmann）发表了自己的观点，然而他的观点并未得到太多关注。注让这种观点重见天日的是查尔斯·桑德斯·皮尔士和理查德·戴德金（Richard Dedekind），朱塞佩·皮亚诺（Giuseppe Peano）将其提炼为皮亚诺公理。

其中的一个基本概念是后继函数，我们用S来表示。一旦你定义了0和后继函数S，就可以将1定义为0的后继，即$S(0)$。你可以用"$2 = S(1) = S(S(0))$"这个等式来定义2。借助这种方法，每个自然数都可以由前一个定义。例如，一旦你定义了0，1，2，3，4，就能用"$5 = S(4)$"来定义5。在定义4时，$4 = S(S(S(S(0)))$，所以$5 = S(S(S(S(S(0)))))$。

在所有情况下，n代表函数S对0应用了n次。一旦你掌握了自然数，就能递归地定义加法。

我使用两个属性来定义“+”（加法）：

$$\text{对所有 } m \in \mathbb{N},\ m + 0 = m$$

$$\text{对所有 } m, n \in \mathbb{N},\ m + S(n) = S(m + n)$$

为了说明这两个属性是如何定义加法的，我们将使用这个定义来分析$3 + 2$。一开始，$2 = S(1)$，所以$3 + 2 = 3 + S(1)$。定义的第二个属性告诉我们，$3 + S(1) = S(3 + 1)$。我们还知道$1 = S(0)$，所以$S(3 + 1) = S(3 + S(0))$。我们再次使用第二个属性，这样就得到了：$3 + S(0) = S(3 + 0)$。因此，我们有了$3+2 = 3+S(1) = S(3+1) = S(3+S(0)) = S(S(3+0))$。现在，我们可以使用第一个属性来推导$3 + 0 = 3$，所以我们得到了$3 + 2 = S(S(3))$。最后，我们使用$4 = S(3)$和$5 = S(4) = S(S(3))$，得到了最后的结论$3 + 2 = 5$。

λ积分和函数

邱奇和他的学生克莱尼、约翰·巴克利·罗瑟（John Barkley Rosser）发明了λ积分来描述计算。他们考虑的对象是函数和基本构造，包含通过替代实现的函数构成的理念。函数是描述计算时的一种合理选择。一个函数需要一个输入，并且能提供一个输出。一个计算需要得到一个函数的输出，并将其作为另一个函数的输入。这就是“通过替代实现的函数构成”的含义。我们将讨论一些例子，帮助大家进一步理解这种方法。首先，我们来描述λ符号的作用。注

恒等函数通常被写作$f(x) = x$。在λ积分中，我们将其写作$\lambda x.x$。点右侧的x与等式右边的x作用一致。λx可以替代x，这就等同于x是一个独立变量或边界变量。与边界变量相关的字母或名称没有实际意义。恒等函数可以写为$f(x) = x$，$g(u) = u$或$h(z) = z$，$\lambda x.x$，$\lambda u.u$，$\lambda z.z$表示的也都是完全相同的意思。有时候，我们会改变一个边界变量的名称以避免歧义。例如，用$(\lambda x.x)(\lambda y.y)$来代替$(\lambda x.x)(\lambda x.x)$。

对于$\lambda xz.xyz$，我们可以替代x和z，但不能替代y。边界变量是x和z。变量y被称作自由变量。被替代的对象出现在表达式的右侧。所以，$(\lambda xz.xyz)a$

意味着一个独立变量被a取代了。规则是从最左侧的边界变量开始向右移动。因此，$(\lambda xz.xyz)a$就变成了$\lambda z.ayz$，$(\lambda xz.xyz)ab$就变成了ayb，$(\lambda xz.xyz)(ab)$就变成了$\lambda z.(ab)yz$。

这种将一个表达式替换为另一个表达式的方式，构成了λ积分的基础。它也许看起来相当简单，但是邱奇、克莱尼和罗瑟发现，他们能够借助这种方法构建自然数和一类更大的函数——可计算函数。

接下来的两节或许看起来很难，第一次阅读本书时，读者可以忽略这部分内容。这种不寻常的思考方式是主要的难点。我们并不习惯将数字视作函数，也不习惯将“是”和“非”视作函数。然而，使用λ积分进行计算相当直接。你可以根据非常简单的规则，将某些符号用其他符号代替。尽管这些概念之后不会再用到，我还是建议读者试着读读看。

算术

λ积分借鉴了一些皮亚诺算术的基本理念。和皮亚诺公理一样，我们拥有0和后继函数。我们定义$1 \equiv S(0)$，$2 \equiv S(1) = S(S(0))$，以此类推。λ积分的不同之处在于我们给出了0和S的定义：

$$0 \equiv \lambda sx.x,$$

$$S \equiv \lambda abc.b(abc)$$

我们将计算一些数字。我们从被定义为1的$S(0)$开始。替换S和0的过程如下：

$$S(0) = (\lambda abc.b(abc))(\lambda sx.x)$$

用$(\lambda sx.x)$替代a，得到$\lambda bc.b((\lambda sx.x)bc)$。

我们考虑表达式$(\lambda sx.x)bc$，并且用s替代b，用x替代c。因为s并没有出现在表达式中，这等同于用x替代c。所以，$(\lambda sx.x)bc = c$，因此，

$\lambda bc.b((\lambda sx.x)bc) = \lambda bc.(b(c))$。重新标记变量后，表达式等于$\lambda sx.(s(x))$。

我们现在拥有了$0 = \lambda sx.x$和$1 = S(0) = \lambda sx.s(x)$。具体来说，就是1是拥有两个输入函数，且将第一个函数应用于第二个函数的函数，即它输入s和x，计算$s(x)$。数字2被定义为$S(1)$。它等于$S(\lambda sx.s(x)) = (\lambda abc.b(abc))(\lambda sx.s(x))$。

至于如何将这个表达式简化为$\lambda sx.s(s(x))$，我想把这部分工作留给读者。数字2是拥有两个输入函数，且将第一个函数应用到第二个函数两次的函数。一般来说，数字n就是拥有两个输入函数，且将第一个函数应用到第二个函数n次的函数，即它输入s和x，计算$s(\dots(s(s(x))))$。

我们拥有了自然数，就可以定义加法。只要使用后继函数，就能实现这个定义。两个数字n和m的和，可以定义为$m + n = mSn$。注

为了解释这个定义，我们将证明$1 + 1$确实等于2。我们以$1 + 1$开始，$1 + 1 = 1S1 = (\lambda sx.s(x))(\lambda abc.b(abc))(\lambda ty.t(y))$。

在第一个表达式中，我们用第二个表达式取代s，第三个表达式取代x。我们就得到了$\lambda abc.b(abc)(\lambda ty.t(y))$。用$(\lambda ty.t(y))$取代$a$，我们就得到了$\lambda bc.b((\lambda ty.t(y))bc)$。

用b取代t，用c取代y，我们就得到了$\lambda bc.b(b(c))$,也就是2（因为我们可以将b变为s，将c变为x）。

从这点来看，你或许赞同哥德尔的观点，认为邱奇的方法给人的直观感受不如图灵的方法清晰，然而，λ积分构建了自身的体系，比图灵的方法更简单一些。为了说明这点，我们将简单描述如何处理逻辑操作。

逻辑

在逻辑学中有两个值，叫作“真”（true）和“假”（false），分别用T和F表示。两个基本的逻辑运算符是“与”和“或”，分别用“∧”和“∨”表示。给定两个描述p和q，非真即假，陈述pq当且仅当p和q都是真时才为真。我们可以将其写作$T \wedge T = T$，$T \wedge F = F$，$F \wedge T = F$，$F \wedge F = F$。

陈述$p \vee q$当且仅当p和q都为假的时候才为假。这可以写作$T \vee T = T$，$T \vee F = T$，$F \vee T = T$，$F \vee F = F$。

在λ积分中，T被定义为$T \equiv \lambda xy.x$，而$F \equiv \lambda xy.y$。因此，T和F都有两个输入表达式。T选择第一个表达式，F选择第二个表达式（善于观察的读者已经

发现了，F的定义与0的定义相同）。

我们来看某些连接。在所有情况下，最左侧的表达式都将接下来的两个表达式作为输入。例如TTF，最左侧的T的输入是T和F。因为T只输出第一个表达式，我们得到$TTF = T$。我们可以直接检查下面这些情况：

$$TTF = T，TFF = F，FTF = F，FFF = F。$$

我们可以通过对比"∧"的真值表来检查答案的正确性。我们定义"∧"是输入两个函数，将它们连接起来，并且在最后增加一个额外的F的函数。所以，在λ积分中，"∧"被定义为$\wedge \equiv \lambda xy.xy(\lambda st.t) = \lambda xy.xyF$。(注)

观察$TTT = T$，$TTF = T$，$FTT = T$，$FTF = F$可以知道，我们可以将"∨"定义为输入两个函数，并且在它们之间放入一个T的函数。因此，"∨"可以被定义为$\vee \equiv \lambda xy.x(\lambda st.s)y = \lambda xy.xTy$。

从这里开始，我们可以构建出所有的函数。令人惊讶的是，这些函数与可以使用图灵机构建出来的可计算函数完全相同。我们现在转向一种完全不同的思考计算的方法。

标签系统

标签系统可以追溯到20世纪20年代波斯特所做的工作。他想将数学中的陈述简化为符号字符串，把证明简化为对字符串的操作。他认为自己最初考虑的是一个简单的例子，但不久后他意识到，事情远非那么简单。

标签系统是一套操作由有限字母表中的字母组成的符号字符串的规则。在每个阶段，字母从字符串首位被去掉，字符串尾端又会加上字母。这个系统被定义为d（我们去掉的字母数量）和与字母相关的字符串。每个字母用自己独特的字符串标记。计算开始时字符串初始字母的标签被添加到字符串的结尾，字符串的头d个字母被删除。如果结果字符串有d个或更多字母，计算会继续。当字母数量少于d个时，计算停止。(注)

举个例子，假设我们的字母表只由字母a和b组成，a被字符串ab标记，b被字符串a标记，我们删除头两个字母（$d = 2$）。如果以bb开始，我们的第一步就是看第一个字母，并且读取它的标签。在这种情况下，它会给我们一个a。我们将a添加到初始字符串的结尾，并且去掉头两个字母。因为字符串里只有一个字母，计算结束。

如果以字符串$bbbb$开始，我们会进行如下的计算过程，得到不断重复的

ab。

$$bbbb$$
$$bba$$
$$aa$$
$$ab$$
$$ab$$
$$ab$$

波斯特思考了由字母*a*和*b*组成的标签系统，其中，*a*由*aa*标记，*b*由*bbab*标记，更新规则是删除头三个字母。他想探讨是否有可能判断出给定一个初始字符串，系统可以获得重复性。他发现这个问题很棘手。明斯基借助计算机来考虑这个问题，也得到了相同的结论。事实上，这个问题仍然悬而未决。

1961年，明斯基证明了标签系统可以模拟出任意图灵机。这意味着给定任意算法，都有一个标签系统可以计算这个算法。这个事实在文字上被表述为标签系统是图灵完备（Turing complete）的。甚至已经有人证明，只需要考虑去掉头两个字母的双标签系统，就可以得到图灵完备性。[注]

给定一个算法，理论上我们可以设计出一个图灵机来实现它，并将其转换为一个标签系统。但是，我们得到的标签系统通常是极端复杂的。构建标签系统前不先设计图灵机，或许是一种更好的选择。这种构想在某些情况下已经实现。我们将探讨一个特定的例子。2007年，丽斯贝特·德·莫尔（Liesbeth De Mol）在自己的博士研究中发现了一个有趣的标签系统。

这是一个非常简单的计算修正柯拉茨函数的系统。

柯拉茨函数是这样的函数：它输入一个正整数，输出另一个正整数。这个函数由两条规则描述，这两条规则取决于输入数字是奇数还是偶数。如果你有一个偶数，那么你就用2去除这个数。如果你有一个奇数，你就用3乘以这个数，然后加1。柯拉茨猜想是如果你以任意正整数开始，持续应用这个函数，你最终会得到1。

我们首先以5来验证这个猜想。因为5是奇数，我们对它乘以3加1，得到16。因为16是一个偶数，我们除以2，得到8。再继续除以2，直到最后，我们得到1。柯拉茨猜想只是一个猜想，没有人能够证明它是对的，也没有人能够举出反例证明它是错的。20世纪60年代有许多数学家研究这一猜想。事实上，众多数学家花费了大量时间来研究这个猜想，以至于有人戏称，设计柯拉茨猜想就是为了拉低美国的数学研究速度。

修正柯拉茨函数基于这样一个事实：将一个奇数乘以3加1，得到的结果一定是偶数。所以，对一个奇数整数应用柯拉茨函数，就能得到一个偶数整数。下一阶段是用2来除这个偶数整数。修正柯拉茨函数一次完成两个步骤。

修正柯拉茨函数$f(n)$的输入是正整数。如果正整数是奇数，就用它乘以3加1，将结果数再除以2。如果正整数是偶数，直接对这个整数除以2。因此，回到我们刚才举的数字5的例子，柯拉茨函数计算后的下一个数值是8（柯拉茨函数从5变为16，再变为8）。递归函数f，得到序列5，8，4，2，最后得到1。

德·莫尔用三个字母a，b，c构建了一个简单的双标签系统。a的标签是bc，b的标签是a，c的标签是aaa。在各个阶段，第一个字母的标签被添加到字符串结尾，然后删除头两个字母。正整数被编码为由全a组成的字符串。a的数量代表了数值。因此，数字5被编码为$aaaaa$。只有完全由a组成的字符串才对应整数。包含b或c的字符串没有对应的数值。它们只对应计算的中间状态。（我在注释中详细介绍了标签系统如何对应修正柯拉茨函数。）

我们还是从数字5开始运行这个系统，5被表示为$aaaaa$。字符串的第一个字母是a。这意味着我们将a的标签bc添加到字符串的结尾，并删除字符串的头两个字母，所以我们得到了$aaabc$。下面的字符串展示了计算的完整过程。只要我们得到一个完全由a组成的字符串，我们就得到了一个整数。在这些例子中，我在左侧一列写下了对应的数字。

因为柯拉茨猜想并未得到证实，德·莫尔设计的这个系统是否对所有字符串都能停止也就不得而知。

5 *aaaaa*

aaabc

abcbc

cbcbc

cbcaaa

caaaaaa

8 *aaaaaaaa*

aaaaaabc

aaaabcbc

aabcbcbc

bcbcbcbc

bcbcbca

bcbcaa

bcaaa

4 *aaaa*

aabc

bcbc

bca

2 *aa*

bc

1 *a*

一维元胞自动机

一维元胞自动机由一个被划分为细胞的无限磁带构成。每个细胞都有一些状态中的一个状态。我们将探讨只有两个状态的情况，这两个状态用白和黑来表示。这意味着这个磁带将被划分为无限数量的细胞，我们将这些细胞描述为方格，每个方格被上色为黑色或白色。

计算以离散时间间隔进行。我们得到的最初的磁带在时间0。它在时间1更新，之后在时间2更新，以此类推。即使它确实是一个在每个时间间隔都会变化的磁带，我们将每个实例描述为一个独立的磁带也更容易一些。给定一个磁带，在一个时间单位后，后续磁带就可以通过一个更新规则来表示，这个规则要求每个细胞查看自己当时的状态和初始磁带上它周围一些细胞的状态，并在更新磁带上给出细胞的新状态。在我们将要探讨的例子中，更新规则将取决于细胞状态和其两侧直接邻接细胞的状态。举个例子，我们可以提出这样一个规则：如果一个细胞和它的两个邻居在时间t有相同的颜色，那么在下一个时刻$t+1$，更新的细胞颜色就将是黑色；否则，下一个时刻这个细胞的颜色就是白色。在这种特定规则下，我们将从$t=0$开始，研究只有三个细胞是黑色的磁带（如图5–1所示）。



图5–1 只有三个细胞是黑色的磁带

在下一阶段（$t=1$），我们将得到如图5–2所示的磁带（你会发现，所有左边和右边的细胞都被涂成了黑色）。

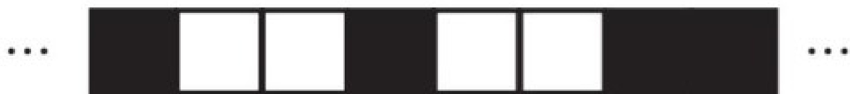


图5–2 t＝1时的磁带

我们画一个网格，这样就更容易看懂发生了什么。第一行是最初给定的磁带。第二行是第一次更新后的磁带，以此类推。在我们的例子中，图5–3的网格展示了最初的磁带和前三次迭代的过程。

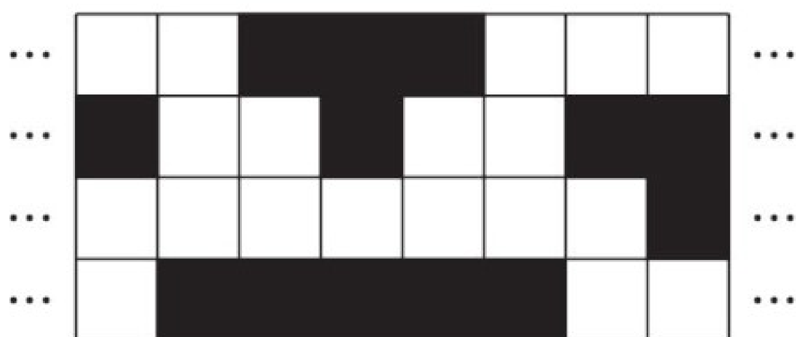


图5–3 磁带迭代网格

图5–3中每个细胞的颜色是通过观察与它直接相邻的细胞确定的。这给了我们一种描述更新函数的方法。三个连续细胞的着色有8种可能性。所有可能性都已经被列举出来（如图5–4所示）。

对每一种可能性，我在图5–5中都画出了下一步被更新的中间细胞的颜色。对我们的例子而言，我们会得到图5–5这样的更新函数。



图5–4 三个连续细胞着色的8种可能性

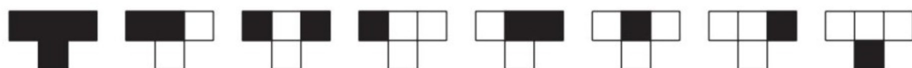


图5–5 更新函数

图5–5为这个规则给出了一种形象表示。为了更新一个细胞，你只需要知道上一个磁带中三个连续细胞的状态。这与图灵机或λ积分中涉及的计算非常不同。例如，在λ积分中，从λ被展开到应用这个概念，仍有一段相当长的距离；对于使用图灵机的计算，磁头通常需要滑动很长一段距离。非常重要的是，元胞自动机在计算上与其他系统等价——任何能由图灵机计算的事物，都可以由一维元胞机来计算。

尽管图5–5是展示此类规则的好方法，我们还有一种更加简明的常用方法。首先，用1表示第二行中的黑色细胞，用0表示白色细胞。图5–5中最左侧的“*T*”有一个黑色细胞，我们用1表示。下一个“*T*”有一个白色细胞，我们用0表示。继续这个过程，我们就得到了10000001，以此精确表示所

有8个“*T*”。然后，我们可以把它想象成一个二进制数字：

$$10000001 = 1 \times 2^7 + 0 \times 2^7 + 0 \times 2^5 + 0 \times 2^4 + 0 \times 2^3 + 0 \times 2^2 + 0 \times 2^1 + 1 \times 2^0 = 128 + 1 = 129$$

对于这些元胞自动机，一共有256条规则。我们已经讨论过的这条叫129号规则。

史蒂芬·沃尔弗拉姆（Stephen Wolfram）在《新科学》（*A New Kind of Science*）一书中对元胞自动机进行了翔实的介绍。我们将要看到的是一个他给出的由元胞自动机完成的简单计算。

考虑一下132号规则。图5–6说明了一些可能的情况。



图5–6 可能的情况

我们将在两个初始磁带上运行它。其中一个有5个连续的黑色细胞，另一个有6个连续的黑色细胞。我们得到图5–7。

给定一个由连续黑色细胞组成的字符串，这条规则将不断减少黑色细胞的数量，直到没有细胞剩余或只有一个剩余。存在两类行为：如果初始字符串中的黑色细胞为奇数个，那么将有一个细胞永远是黑色的；如果初始字符串中的黑色细胞为偶数个，那么最终将不会有黑色细胞剩余。我们可以将这条规则看作判断一个数字是奇数还是偶数。

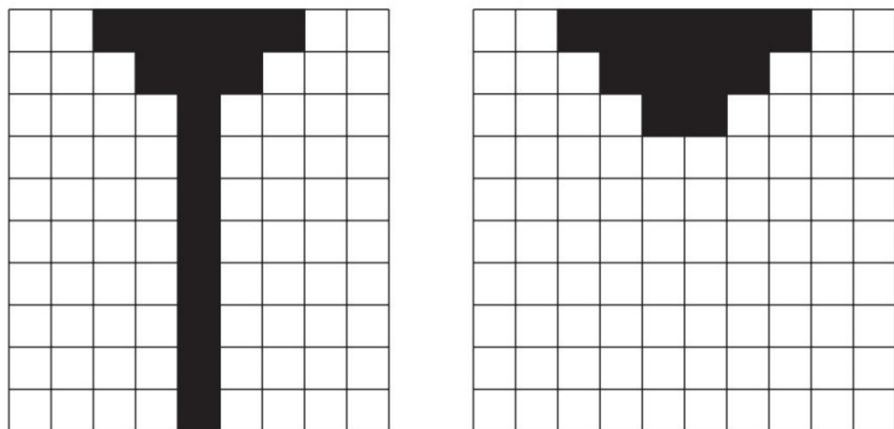


图5–7 运行后的情况

元胞自动机有一个非常好的特点，那就是我们可以画图展示计算的全过程。沃尔弗拉姆在《新科学》中讨论了全部256条规则，并根据发生的事件将这些规则分为4类。第一类规则有非常简单的行为，系统稳定。第二类规则会产生震荡型计算。第三类规则生成查找混沌的计算。第四类规则是最有趣的。这类规则中的计算展现出了混乱和稳定的混合。沃尔弗拉姆猜测，这类规则是可以进行任何计算的通用机器。在下一章，我们将探讨通用机器，并且会提到马修·库克（Matthew Cook）证明110号规则是通用的。注 如此简单的事物能够进行任何计算，这本身就是一个令人惊讶的结果。

---

1. 这个机器有所不同。其中两个版本源自巴贝奇的设计。一个在伦敦科学博物馆展出，另一个由内森·梅尔沃德（Nathan Myhrvold）所有，在加州计算机历史博物馆展出。
2. 查尔斯·巴贝奇发明了分析引擎，都灵大学军事工程负责人梅纳布雷亚（L. F. Menabrea）绘制了分析引擎的草稿，洛芙莱斯伯爵撰写了回忆录笔记。
3. 大卫·多伊奇，《无限的开端：改变世界的解释》。
4. 这些出现在《艾伦·图灵的逻辑系统：普林斯顿论点》的序言中。在苹果公司的网站上同样可以看到这些内容。
5. 赫尔曼·格拉斯曼的《Lehrbuch der Arithmetik für höhere Lehranstalten》一书可以在互联网档案馆找到。

6. λ积分中的λ最初源自罗素和怀特海使用的标记。他们使用了$\hat{x}$。邱奇感觉符号ˆ应该写在x之前，即ˆx。这个符号随后被排版为λx。

7. 函数“＋”需要两个数字作为输入，并给出一个输出。在λ积分中，＋通常被写在第一位。因此，应该写成$+(m)(n)$，尽管看起来奇怪，却让人更清楚＋是有两个输入的函数。如果使用这个标记，那么＋就可以被定义为$+ \equiv \lambda mnsx.ms(nsx)$。

8. 和＋一样，也是有两个输入的函数。相比于写成$T\wedge F$，应该写成 $\wedge(T)(F)$。

9. 波斯特使用“标签”这个术语，是因为这个从头“跑”到尾给每个字母标记的过程，让他想起了儿时的标签游戏。

10. 马文·明斯基对这些系统进行了大量研究。他的著作《计算：有限和无限机器》是一部杰出的作品。

11. 丽斯贝特·德·摩根（Liesbeth De Mol），《标签系统和类似坷拉兹函数》。《理论计算机科学》，390卷，2008年，第92～101页。

12. 以数字n开始的计算可以表示为n个a的字符串。当我们应用标签系统时，a被b和c取代。在第一阶段，我们从字符串前面去掉aa，在字符串尾部加上bc。因此，字符串的长度一直是n。根据初始数字n是奇数或偶数，有两种情况需要考虑。

13. 马修·库克，《元胞自动机的一般性》，《复杂系统》，2004年。

第六章 编码和通用机器

给你纸、笔、橡皮以及规则条件，你实际上就是一台通用机器。

——艾伦·图灵

在讨论过多种形式的计算后，我们再回到图灵的论文上来。我们已经知道，图灵需要为算法做出一个正式的定义，他实现这个定义的方式，正是我们现在所说的图灵机。邱奇—图灵论题认为，对于任意一个算法，有可能设计出一个能够执行该算法的图灵机。

因为图灵机被称作机器，我们倾向于认为图灵机是物理机器。事实上，给定一个图灵机的具体例子和一个磁带，我几乎可以想象得到它每一次改变状态时发出的微弱的“嘀嗒”声，随后是进入接纳状态时令人神往的“嘡”的一声。然而，关于这些机器最重要的事实是它们是在描述算法。图灵发现了一种对算法进行完整定义的优雅方案。

当你得到一段描述后，你也许会用毫不相同的词汇描述完全相同的事物。这就是图灵下一步要做的事。他拿到自己的机器，并且证明图灵机的图给出的所有信息都可以被编码为由0和1组成的字符串。现在，你可以用一串二进制数字来表示图灵机的图描述的算法。

如果你拥有一些编程经验，就会知道程序通常是使用英文词汇、以高级语言编写的，但是运行程序需要计算机将其转换为由0和1组成的字符串。我们可以将图灵机视作程序。这样人们就很自然地尝试将图灵机的图转换为被我们称为编码的二进制字符串，这也是我们在这一章中将要关注的主题。我们将看到，编码自然而然地引出通用计算机的概念。在后续的章节中，我们将借助编码和著名的康托尔对角论证法来展示计算的瓶颈。

将算法转换为二进制字符串的想法，在今天看来也许非常自然，但在图灵的年代却并非如此。图灵从科特·哥德尔的研究中得到了编码图灵机的灵感——后者将很多数学中的陈述编码为数字字符串。哥德尔采用的数字化方法在逻辑上十分有效，但在数学的其他领域并未被广泛使用。图灵借用了这一概念，并将其应用到算法上。在这个过程中，这一理念逐渐成为奠定现代计算机基石的基本概念之一。图灵是认识到算法和数据可以被编码为单一数字字符串的第一人。

编码有限自动机的方法

有限自动机比图灵机更简单、更容易编码。这意味着将图转换为字符串的过程会更快，结果字符串也更短。尽管有限自动机比图灵机简单，我们也可以借助完全相同的理念进行编码。一旦你了解如何编码一个机器，就能够设计出一种可以编码任何机器的方法。

我们假设数字字母表只包含0和1（如果输入字符串使用了不同的字母表，应该将其转变为只使用0和1的字母表）。

为了对有限自动机进行完整的描述，你应该列出如下信息：

1. 状态数。

2. 开始状态。

3. 接纳状态。

4. 对于每个状态，如果输入0会发生什么，如果输入1会发生什么。

为了便于描述，我们假设所有状态都已经被编号，并且第一个状态就是开始状态。

现在，我们需要一种方法将描述编码为二进制字符串。关键是使用0来编码数字，使用1来编码其他信息。这里的其他信息类似于标点符号。用文字来表述就是你需要用标点符号告诉我们，句子从哪里开始，到哪里结束，并以此来区分主从句。在描述机器时，我们需要做相同的事情。我们将在编码的开始和结束部分使用4个连续的1。3个连续的1表示一个种类的停止——类似于句号。两个连续的1表示子类的改变——类似于分号。一个1的用法则类似于逗号。下面这个具体例子会让各位读者有更加清晰的认识。让我们考虑一下之前探讨过的M_2机器。如图6–1所示，二者唯一的不同是状态的名称。在前面的讨论中，我们用字母来标记这些状态，现在我们将使用连续整数来标记，开始状态标记为1。

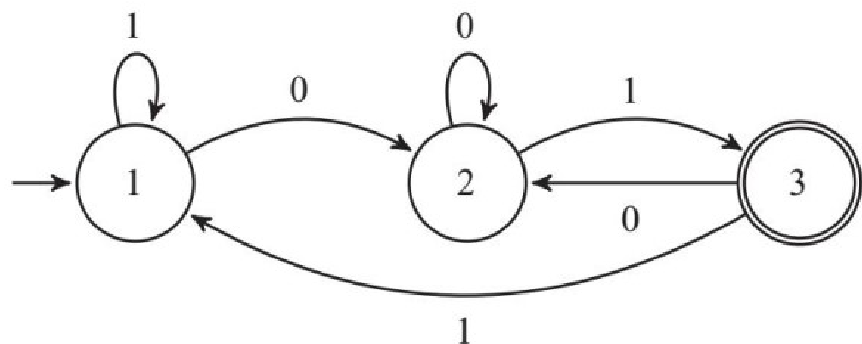


图6–1 有限自动机

编码的开始和结束都用4个1来表示。所以，编码的前4个符号是1111。我们接下来需要输入状态的数量。这个机器有三个状态，我们用三个0来表示这些状态。所以编码变成了1111000。我们已经讨论完状态的数量，所以我们在结尾加上111。下一个要输入的是输入状态列表。在这里，只有状态3能被接纳，我们用三个0来表示这个状态。我们的编码现在变成了1111000111000。我们再输入三个1来表示已经讨论完接纳状态，这样一来，我们就得到了1111000111000111。

如果状态2和状态3都成为接纳状态，我们就可以将这种信息编码为0011000。两个1分隔了对状态的描述。如果三个状态都是接纳状态，我们就可以将这种情况写作0110011000。

现在，我们需要给出信息，描述当我们处于一个状态并且输入0或1时，会发生什么。我们轮流进入各个状态。对于每个状态，我们首先要明确，当输入为0时进入哪个状态，当输入为1时进入哪个状态——使用一个1来分隔这些信息。

在状态1中，如果输入0，我们移动到状态2，如果输入1，我们移动到状态1。这被编码为0010。在状态2中，如果输入0，我们移动到状态2，如果输入1，我们移动到状态3。这被编码为001000。在状态3中，如果输入0，我们移动到状态2，如果输入1，我们移动到状态1。这被编码为0010。这三段信息都被两个1分隔开。至此，我们得到了0010110010001100 10。将它添加到我们刚才的编码后，就得到了111100011100011100101100100011001 0。最后，我们添加4个1，表示我们已经完成了描述。这样，我们得到的编码就是

111100011100011100101100100011001011111。

一个机器编码的标准标记是〈M〉。所以，〈M_2〉=111100 011100011100101100100011001011111。

我们不仅能编码有限自动机，获得一个由0和1组成的字符串，还能解码这个字符串。我们不仅能从M来到〈M〉，而且能从〈M〉回到M。使用我们的解码方法就能够实现。为了进行说明，让我们再来看看1111001110011100101101001111这个字符串。我们将逐步进行解码。

前4个1说明这是对一个机器描述的开始，最后4个1说明我们正在结束描述。现在，我们读入第一个由0组成的子串

1111001110011100101101001111

这里加粗的两个0告诉我们，这个机器有两个状态。下一个由0组成的子串

1111001110011100101101001111

告诉我们，状态2是接纳状态。

再下一个加粗的子串

1111001110011100101101001111

告诉我们，如果在状态1接收了0，我们将移动到状态2，如果接收了1，我们将移动到状态1。最后这段被加粗的子串

1111001110011100101101001111

告诉我们，如果在状态2接收了0，我们将移动到状态1，如果接收了1，我们将移动到状态2。这是对有限自动机的完整描述。图6–2画出了这个有限自动机。这个机器是用来识别拥有奇数个0的字符串的有限自动机。

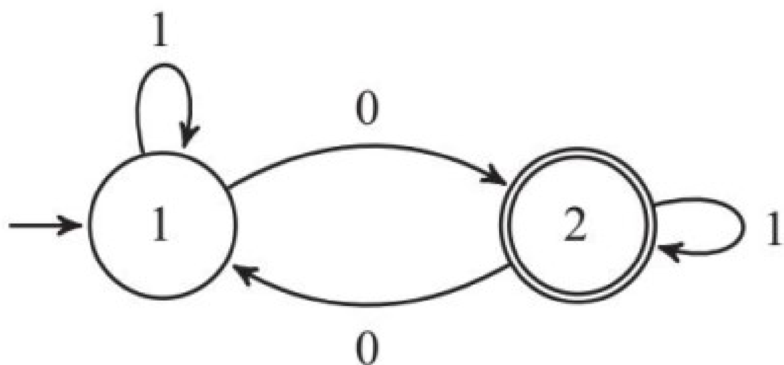


图6–2 有限自动机

通用机器

如果我们要进行实际运算，就需要给有限自动机加上一个输入字符串。为了将这个信息整合进编码中，需要在编码最后进行一些处理。如果给定机器M和输入字符串I，我们将用〈M，I〉来表示字符串〈M〉和输入字符串I的组合。假设有人给我们提供了如下字符串，并且想要知道计算的答案是否被接纳。

〈M，I〉 = 111100111011100101101001111110010110

我们知道，对机器的编码会在输入字符串前被给出。我们还知道，机器编码以4个1开始。所以，当我们再次读入4个1的时候，我们知道编码已经结束，输入字符串将要开始。在这个例子中，我们得到了〈M〉 = 1111001110111001011010011111和I = 110010110

我们现在可以解码机器的编码，并且画出这个机器。我在图6–3中完成了这项工作。它与之前的机器一样，不同之处在于第一个状态取代第二个状态成为接纳状态。

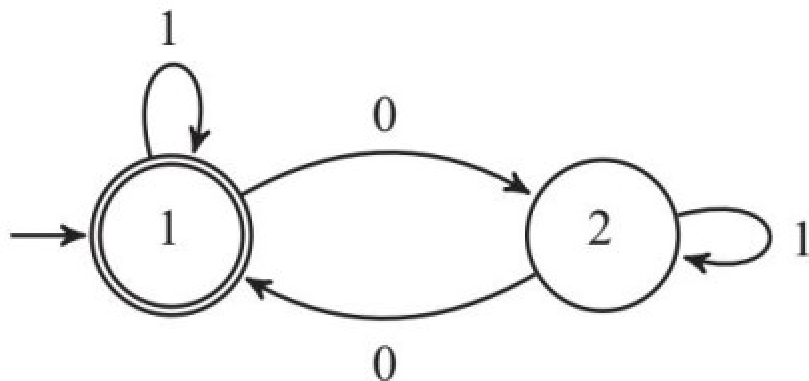


图6–3 有限自动机

这个机器接纳拥有偶数个0的字符串。我们输入的字符串是110010110，拥有4个0，所以机器可以接纳它。因此，

〈M，I〉 = 1111001110111001011010011111110010110

这个计算可以被接纳。

我们了解了如何拿到有限自动机M和输入数据I，将这两段信息编码为一个字符串〈M，I〉。我们也知道了如何逆转这一过程。给定字符串〈M，I〉，我们有办法将〈M〉和I分隔开，之后再重构原本的自动机M，最后在我们重构出来的机器M上运行字符串I。

这样来实现我们的目的，看起来似乎太过迂回，然而这其中蕴含着一个重要的理念。让我重述上一段最后一句，再做一点改动：给定字符串〈M，I〉，我们得到了将给定字符串〈M〉与I分隔开的算法，之后重构原本的自动机M，最后在我们重构的机器M上运行字符串I。根据邱奇—图灵论题，如果我们有一个算法，就可以设计一个图灵机来执行这个算法。因此，会有一个图灵机能够接纳〈M，I〉作为输入，模拟机器M接收输入I的过程。结论是有一个图灵机能够模拟运行任何输入的任意有限自动机。很明显，这是一个强大的机器，但是我们能制造出一些更强大的东西。

我们已经讨论了有限自动机，我们可以使用几乎一样的方式来设计图灵机。两者唯一的区别在于，图灵机更复杂，需要更多信息进行描述，这也就产生了更长的编码。像处理有限自动机那样，有一种将图灵机从编码中重构的算法。

如果给定一个编码〈M，I〉，其中M为图灵机，I为输入，我们知道有一个

算法能够重构M，并且在重构出的M上运行I。这句话也可以重新表述为，给定一个编码〈M，I〉，有一个算法将〈M，I〉作为输入，如果M接纳I，将给出“接纳”这个答案，如果M拒绝I，将给出“拒绝”这个答案，并且如果M没有在I上停止，整个过程就不会停止。

现在，根据邱奇—图灵论题，我们知道每个算法都存在一个对应的图灵机。所以我们得到：存在一个图灵机U，将接受任意〈M，I〉作为输入，如果M接纳I，给出“接纳”这个答案，如果M拒绝I，给出“拒绝”这个答案，并且如果M没有在I上停止，整个过程就不会停止。

这个机器U能够在任意输入上模拟任意图灵机。能够完成这个行为的机器被称作通用图灵机。

现代计算机就是通用图灵机。我们可以将M视作程序，将I视作数据。〈M，I〉就对应转换为二进制字符串的程序和数据。当计算机运行〈M，I〉的时候，它就是在程序M上模拟输入I。

设计通用机器

通用机器是一台计算机，它能够读取由图灵机和输入字符串组成的描述，并在图灵机上模拟这个字符串的运行。首先，让我们简单地思考一下人类和人类的能力。

在介绍图灵机的章节中，我们讨论了一个能够查看由0和1组成的字符串，并且判断0和1的数量是否相等的图灵机。之后，我们又讨论了一个运行字符串001的图灵机的例子。

对图灵机的描述是由图给出的。通过这个描述和输入字符串011，我们可以模拟机器的运行。我们或者完全在大脑中实现这一过程，或者在纸上写出中间步骤，但无论是哪种情况，我们的大脑都会模拟这个机器在运行自己的输入。在上一章中，我们看到给定任意图和一个输入字符串，我们就可以在这个输入上模拟运行这个机器。当时我们以为我们在研究图灵机如何工作，然而在这个过程中，我们看到了我们拥有在任意输入上模拟任意图灵机的能力。这种能力意味着我们就是通用图灵机。我们拥有运行任何算法的能力。

“我们是通用图灵机”这个说法好像在颂扬我们的能力，但是事实并非如此。如果我们得到了一组告诉我们在所有情况下应该做出什么反应的指令，我们就拥有执行这个指令组的能力。一旦你理解了图灵机的图在描述什么，就很容易在一个图灵机上运行一个输入字符串。这几乎不需要任何脑力参与。在每个阶段，只有一件接下来需要完成的事情。因此，通用图

灵机不必是复杂的机器，它可以是相对简单的事物。

在上一节中，我们讨论了一个通用图灵机U将〈M，I〉作为输入，并且在输入I上运行M。机器U可以相当简单，M则更加复杂。换言之，设计一个算法要比设计一个运行这一算法的机器困难。

你或许想知道，是否有可能画出一个通用图灵机的图。答案是肯定的，然而想要解释清楚这个机器的每一个状态都在做什么是一个相当宏伟的工程。

我们不会过多地探讨设计一个通用图灵机的细节，但对于想要深入了解的读者，我建议阅读马文·明斯基的著作《计算：有限和无限机器》，其中提到了不少例子。在状态数量和磁带上允许的不同符号的数量，以及使整个系统运行的复杂性之间，需要认真权衡。他提出的第一个例子具有23个状态和6种磁带符号。这是最容易理解的例子，状态被分类为能够实现不同功能的部分。他给出了一个只包含7个状态和4种符号的例子，但是他也指出："正如读者将要看到的，这个机器或其他类似机器将不适合用于解释说明。"

相比于显式设计一个通用图灵机，我们将稍稍走一点弯路。我们将先证明现代计算机是通用机器，之后再证明它们能够用图灵机来表示。

这种方法的优点在于它令邱奇—图灵论题看起来更加可信。这个论题提到，给定任意算法，总能设计出图灵机来执行这一算法。一旦我们证明了现代计算机和图灵机在计算上是等价的，这个论题就可以重新表述为：给定任意算法，总能设计出一个现代计算机来计算它。这似乎更令人信服。

现代计算机是图灵机

为了证明计算机是通用图灵机，我们必须证明计算机可以在任意磁带上模拟任意图灵机。这个想法看起来似乎非常合理，我们将规划出一个框架来分析究竟如何证明这个想法。

对人类而言，拿到描述图灵机的图是有帮助的，然而对计算机而言，最好不要用图来描述图灵机。幸运的是，我们很容易做到这一点。

当我们介绍图灵机的时候，我们从图6–4开始。

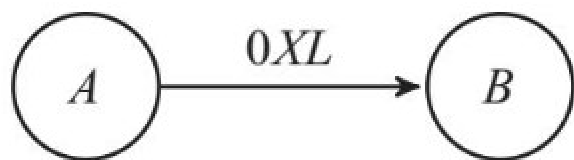


图6–4 图灵机

图6–4告诉我们，如果处在状态A，并且接收了输入0，那么下一步就是移动到状态B，在磁带上写下X，将磁头向左移动。我们可以用一个五元组(A, 0, B, X, L) 来表示相同的信息：第一个是当前状态，第二个是当前磁带符号，第三个是新状态，第四个告诉我们在磁带上写入什么，最后一个告诉我们应该将磁头向哪个方向移动（这种使用五元组的方法实际上就是标准标记法，图灵并没有画图，他使用了五元组）。

一旦你用五元组来描述图灵机，就很容易编写出一个计算机程序来模拟图灵机在输入磁带上的活动。程序读入五元组和输入字符串。在每个阶段，程序读入当前状态和当前输入符号。随后，它搜索所有五元组，查找到前两个条目符合的五元组。再之后，它用另外三个条目更新状态，更新存储字符串，并选择下一个输入符号。

这个论证相当简单，它给出了“如何在计算机上模拟运行任意输入的任意图灵机”的大致过程。它向我们展示了现代计算机是通用机器。

这样一来，我们能够在现代计算机上模拟图灵机，就不再那么令人惊讶了。真正令人惊讶的是我们能够设计出一个图灵机来模拟现代计算机，这说明图灵机在计算能力上是与现代计算机等价的。我们将描述如何实现这个过程。

冯·诺依曼结构

《关于EDVAC的报告草案》或许是关于现代计算机设计的最为重要的论文。这篇论文写于1945年，当时人类正在制造第一批电子计算机。这篇论文描述了计算机设计的基本框架，融合了前人研究的设计理念。

这篇论文的视角与图灵论文的视角截然不同。图灵感兴趣的是究竟什么是可以计算的。他的机器只是整合了人类计算者基本计算步骤的理论结构，而冯·诺依曼感兴趣的是设计一台有效的机器来运行实际计算。这种想法带来的设计就是人们常说的冯·诺依曼结构，大多数现代计算机都是基于这种结构设计的。

冯·诺依曼的设计汲取了许多人的理念。《EDVAC的报告草案》只是一篇论文的草稿，原本只是给一小部分人传阅的。如果像原本计划的那样，这篇论文只在小范围内传阅，“冯·诺依曼被列为唯一作者，其他人的研究成果并没有被提及”这一事实就不会造成太大的影响。然而，这篇论文很快被传播开来，并且对后续的计算机设计产生了巨大的影响。这篇论文中一些概念的发起者，比如普雷斯伯·埃克特（J. Presper Eckert）和约翰·莫奇利（John Mauchly），对自己的研究在冯·诺依曼结构中起到了关键性作用但并未被提及，感到相当沮丧。

计算机的心脏是CPU（中央处理器）。CPU从内存中载入指令，进行解码和执行。这可能包含从内存读入更多数据或执行简单的结构和逻辑操作。CPU由一个控制单元和一个算术逻辑单元组成。这些单元都包含特定的快速内存单元，即寄存器。一个特殊的寄存器是程序计数器。它的作用是记录下一条要执行的指令的地址。在每个周期开始时，控制单元从计算机中读入下一条指令的地址，并使用这个地址从内存单元载入指令。

这些指令列出了CPU的最基本操作，包括在内存和寄存器之间移动数据的规则，以及使用算术逻辑单元完成的一些简单操作。任何计算机能够运行的程序都必须转变为这些基本机器指令的组合。我们一般会使用高级语言编写程序，但是在计算机实现这些程序前，必须使用编译器将这些程序转换为机器语言。

在判断哪些东西理论上可以在现代计算机上计算之前，我们需要一个数学模型来涵盖所有必要的特征。随机存取机器（Random Access Machine，简称RAM），就是这样的一个数学模型。

随机存取机器

随机存取机器是一个具备冯·诺依曼结构的理论机器模型。这个模型虽然是理论上的，但却拥有具备这种结构的实际计算的所有基本性质，而且它是一个描述大多数现代计算机的优秀模型。注

RAM这三个首字母缩写有两重含义：一个流行的含义是随机存取内存；另一个不太常见的含义就是随机存取机器。所有计算机都会在内存中存储一些东西。每个东西都会有一个属于自己的地址，这个地址告诉计算机数据被存储在内存中的什么位置。如果内存是随机存取的，那么每个地址都可以被即时定位。计算机不必搜索整个地址列表来找出一个特定的地址。

图灵机使用磁带来存储信息，并不具有随机存取能力。如果机器需要使用的信息被存储在距离磁头当前读取位置很远的单元中，那么磁头必须经过

所有中间单元，才能移动到指定位置。如果随机存取机器需要某个信息，就能够即时来到这个信息所存储的位置。

RAM拥有无限多个内存单元，分别用M(1), M(2), M(3)表示；以及有限个寄存器，分别用R_0, R_1, ..., R_r表示。每个内存单元和寄存器都可以存储一个整数。还有一个名为程序计数器的东西能够存储非负整数值。

这个机器运行一个由一系列指令组成的程序，使用程序计数器为程序提供下一条待执行指令的编号。每条指令告诉寄存器R_0和程序计数器应该如何更新内容。因为指令列表相当短，我们将完整地列出来。

前三个指令包括将数字载入寄存器。我们可以将一个整数n载入R_0，将存储在任意寄存器中的数字载入R_0，将存储在R_0中的数字载入任意寄存器。我们分别用三个表达式表示这三个操作：

$$R_0 := n,\ R_0 := R_s \text{ 以及 } R_s := R_0$$

我们可以做一些简单的算数操作。我们将一个数字n加到存储在寄存器R_0的数值上，将任意寄存器中的数字加到存储在寄存器R_0的数值上，将任意寄存器中的数字乘到存储在寄存器R_0的数值上。我们用下列表达式来表示这些操作：

$$R_0 := R_0 + n,\ \ R_0 := R_0 + R_s \text{ 以及 } R_0 := R_0 \times R_s$$

我们还可以将存储在任意内存地址的数字载入R_0，或者将R_0中的数值写入任意内存地址。这些操作可以写作：

$$R_0 := M(R_s) \text{ 和 } M(R_s) := R_0$$

在这里，$M(R_s)$ 表示首先读入存储在R_s中的整数，随后将其作为内存地址。注

程序计数器通常在一条指令被读取后增加1。这意味着机器在读入一条指令后，会来到列表上的下一条指令，但是我们有很多方法来修改程序计数

器。这个程序可以调到第n条指令，我们将其表示为

$$\mathrm{PC:} = \mathrm{n}$$

我们也可以使用两个条件论述，包括存储在R_0中的数字是否等于0。

$$如果R_0 = 0，那么PC: = n,$$

以及

$$如果R_0 > 0，那么PC: = n$$

当我们开始一个计算的时候，存储在寄存器和内存中的数字都将被清零。输入包括一个数字字符串，代表程序和被读入内存的数据。如果数字字符串是k个单位长，它就会被读入$M(1), M(2), \ldots, M(k)$ ，其余的内存单元仍然为0。程序计数器设置为1。

RAM为我们提供了一个现代计算机的出色模型。从最基本的层面来看，现代计算机完成的是基于一些简单操作的计算，而且这些计算实际上就是RAM描述中列出的操作。因此，为了证明图灵机能够模拟现代计算机，模拟一个RAM就已经足够。

图灵机能够模拟RAM

设计图灵机来实现整数加法和乘法相当简单。一个加法机将输入整数m和n，在计算结束时将整数$m + n$留在磁带上。一旦我们拥有了可以进行加法和乘法的图灵机，就可以将它们整合为一个同时具有加法和乘法能力的图灵机。和编程中使用的子程序一样，我们能够将可以实现简单功能的图灵机整合为一个需要使用这些简单功能的更加复杂的图灵机。一个模拟RAM的图灵机需要具有加法和乘法功能。我们将假设我们已经设计出可以完成这些操作的图灵机，并且可以将它们整合到我们的新机器中。

我们无法通过赋予图灵机更多功能来让它们变得更强大。具体来说，即使为它们提供更多磁带和磁头，也不能提升计算能力。通过证明任意具备多磁带的图灵机可以用具备单一磁带的图灵机模拟[注]，我们可以证明上述观点。将RAM转换为多磁带图灵机将是我们的第一步。完成了这一步，我们就可以将其转换为单一磁带图灵机。

我们最初的图灵机只有一个用于内存的磁带，一个用于寄存器和程序计数器的磁带，一个额外的用于草稿的磁带。首先，我们来描述一下内存磁带。

RAM开始输入一个列出了数据和指令的字符串。我们用$v_1v_2...v_k$来表示这个字符串。图灵机需要记录每个符号在字符串中的位置——即它需要记录下标。为了实现这一点，我们将在字母表中增加两个新符号，分别用# 和 * 来表示。

之后，我们用输入字符串$v_1v_2...v_k$来初始化我们的图灵机并且插入符号，这样一来，我们就在内存磁带上得到了：

$$\#1*v_1\#2*v_2\#3*v_3\#\cdots\#k\ *v_k\#$$

这确保了我们可以记录数字对应的每条指令。如果我们要找原始RAM输入字符串的第二个元素，可以先找新字符串中的#2*。一旦看到这个，我们就知道v_2将要成为下一个字符串。我们知道，在读入#2*后，字符串v_2立即开始，并且在我们读入下一个#前结束。

其他磁带最初都是空白的，只有程序计数器的磁带初始值为1。

我们将给出一个图灵机的例子。假设它已经在运行，现在我们观察一下在完成9条指令后的情况。我们将看到它如何处理第10条指令。在完成一条指令后，这个机器会从程序计数器磁带上读入待执行的下一条指令的编号。我们假定第9条指令并未涉及跳转，而且程序计数器每一个阶段只更新一个数值。之后，程序计数器将写上10。机器搜索整个内存磁带来寻找#10。它找到了#10 * v_{10}# 。接下来，它要检查v_{10}。这或许是对应一条指令的一个整数。让我们假设，它对应指令$R_0 := M(R_3)$ 。这个机器随后读入R_3磁带上的整数，并在内存磁带上寻找这个整数。例如，假设R_3包含整数41。机器会在内存磁带上寻找#41*。它找到#41* v_{41}# 并将v_{41}复制到磁带R_0。最后，它在程序计数器上加1。第10条指令完成，开始第11条指令。

所有RAM的机器指令都可以在具备多磁带的图灵机上完成。具备多磁带的机器可以由单一磁带的图灵机模拟。当然，这个新的图灵机将拥有许多状态，并且相当复杂。执行一条简单的RAM指令需要很多个步骤。但是，我们对如何真正构建这个机器并不感兴趣。我们只是想证明这个转换理论上是可行的。一旦我们确信可以用图灵机模拟RAM的所有基本指令，我们就可以模拟整个RAM。这就说明图灵机具备现代计算机的计算能力。

其他通用机器

我们已经看到现代计算机是通用机器，一些非常简单的计算系统也是通用的。明斯基研究了波斯特的标签系统，并且能够构建出一个模拟通用图灵机的标签系统。因此从理论上说，标签系统能够完成图灵机或计算机能完成的工作。

在讨论一维元胞自动机的时候，我们观察到一些能够被视作计算算法的规则。令人惊讶的是，它们也可以模拟通用图灵机。史蒂芬·沃尔弗拉姆猜测，规则110（如图6–5所示）是图灵完备的。这是一个依赖开始磁带并展现出混乱和稳定混合的规则。对一些开始磁带而言，这条规则产生了非常简单的输出。对其他开始磁带而言，输出看起来是杂乱无章的。沃尔弗拉姆猜测规则110应该能够进行任何计算。



图6–5 规则110

马修·库克供职于沃尔弗拉姆研究中心，他成功地证明了沃尔弗拉姆的猜测是正确的。这是一个令人印象深刻的结论。给定一个计算，设计一个开始磁带让规则110执行它并没有那么容易。库克是如何证明的？第一步是借助一个事实：图灵机能够由某类标签系统模拟。之后，他证明了这些标签系统能够由规则110模拟。

当然，这是一个理论结论。很难想象一台实际计算机能够用这条规则来设计。但是值得注意的是，某些很简单的东西也可以是通用图灵机。

一般而言通用计算机并不复杂，复杂的是编程。在规则110这个例子中，我们能够通过正确选择初始磁带配置来证明它可以模拟任意元胞自动机。虽然从实践上来看，从一个算法到设计正确的初始磁带没有捷径可走，但重要的是这一过程并非无法实现。我们知道，对任意算法而言，一定有一个元胞自动机可以实现这个算法，一定有一些用于规则110的输入磁带可以模拟这个自动机的计算。

当我们把〈M〉输入M的时候会发生什么

给定一个机器M，我们可以把它编码为字符串〈M〉，可以设计一个以〈M，I〉为输入的通用图灵机U，并且模拟输入I运行M。如果让一个机器运行自己的编码会发生什么？我们可以探讨可能发生这种情况的一个实际例子。

$$\langle M_9 \rangle = 1111001110011100101101001111$$

这个机器在图6–2中进行了描述，它接受包含奇数个0的字符串，拒绝包含偶数个0的字符串。编码〈M_9〉有10个0，所以如果我们在〈M_9〉上运行〈M_9〉，机器将会拒绝它。这就等同于如果我们把〈M_9，〈M_9〉〉输入U，它将以拒绝状态停止。

$$\langle M_9, \langle M_9 \rangle\rangle =$$

$$11110011100111001011010011111111001110011100101101001111$$

接下来，让我们考虑一下M_2。这个机器接受结尾是01的字符串。我们知道，任意一个机器的编码结尾都是4个1。所以〈M_2〉以4个1结尾，将会被M_2拒绝。

最后，让我们考虑一下M_3。图6–6重新画出了这个图灵机。这个机器接受由偶数个1组成的字符串。我们得到编码

$$\langle M_3 \rangle = 1111001110111010011001011111。$$

因此，这个编码有偶数个1，M_3将会接纳它。

正如我们将会看到的那样，这种自我参考的理念将被证明有重要意义。让我们先看一个实际的例子，了解为什么我们可能会在一个机器（或算法）上运行它自己的编码。注

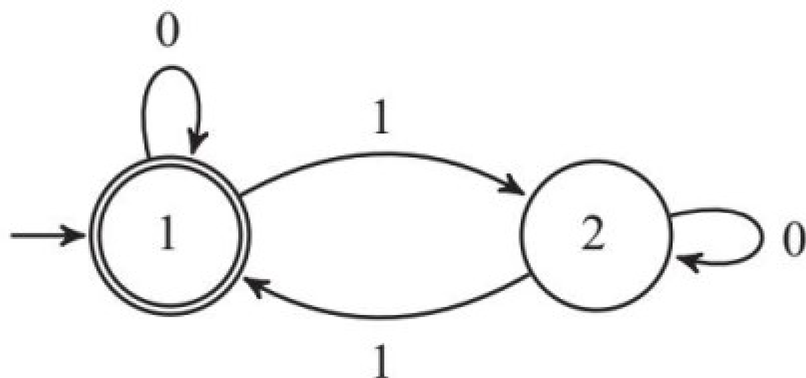


图6–6 图灵机

我们使用高级语言编写程序，但是实际运行时必须先将这些程序转化为机器语言。这种转化是由编译器完成的。假设我们当前的编译器并不是非常有效，我们编写一个能够将编译程序所需运行时间缩至最短的新编译器。我们用*Q*来表示经过优化的编译器。因为我们用高级语言编写了*Q*，在计算机能够执行以前，必须在当前的编译器上运行这个新编译器。我们用*Q*′来表示经过编译的新编译器。我们想要一个经过优化和编译的*Q*。所以，我们应该使用*Q*′来编译*Q*。这会让我们得到经过优化和编译的*Q*，我们用*Q*″来表示它。现在，假设我们以这种方式设计*Q*，并且在之后优化机器代码，我们将在*Q*″上运行*Q*″，以便得到经过双重优化和编译的*Q*。注

从这一点来看，可能我们还是不清楚为什么在一个机器上运行它自己的编码是重要的，但是我们不久将看到，正是这种理念促成了理论计算中一些最令人不可思议的、最精妙的结论。

---

1. 1973年，史蒂夫·库克和罗伯特·雷金浩（Robert Reckhow）最早介绍了这些机器。

2. 这需要R_S中的整数是非负数。

3. 转换中的关键思想就是处理磁带上的信息。每个磁带上的信息都被复制到拥有一个磁带的机器上。一个新符号，比如#，被用来分割这些磁带的复制品。我们必须让每个磁带拥有自己的头部。完成这一点的方式是在字母表中拿到所有字母，对每个字母稍加改动。如果0在字母表中，我们可能会发明[image]。这个新符号让机器知道磁头位于什么位置。例如，如果我们有两个磁带，第一个磁带上写着*abbaaa*，磁头位于第一个*b*上方，第二个磁头上写着010101，磁头位于第二个0上方。我们会将这个信息表示为

$$\#a\hat{b}baaa\#01\hat{0}101\#$$

其余的转换相当枯燥，但是重要之处在于这项工作可以完成。

4. 明斯基的著作《计算：有限和无限机器》包含大量关于标签系统的信息。

5. 库克的表示法在指数时间内模拟了图灵机。特洛·内亚里（Turlogh Neary）和达米安·伍兹（Damien Woods）在2006年使用110号规则给出了另外一种表示图灵机的方法，只需要多项式时间。这意味着，任何图灵机能够在指数时间内解决的问题，都可以使用110号规则在多项式时间内解

决。

6. 兰道尔·门罗（Randall Munroe）创立了网络漫画网站xkcd。漫画《一堆岩石》揭示了110号规则和宇宙的出现。

7. 作者想要感谢一位匿名推荐人建议使用这个例子。

8. 真实世界的编译器通常由它们编译的语言编写。这个过程被称作引导程序。一种新语言的第一个编译器通常是使用其他语言编写的，但是后续的编译器通常以被编译的语言写成。这引发了一个有趣的问题：你有多大把握让自己程序的编译版本正确运行？

第七章 不可判定的问题

现在，我们回到判定问题。判定问题是一个取决于输入参数的问题。一旦我们为输入参数赋值，它就变成了一个是非问题。如果存在一个算法能够在所有情况下给出正确答案，那么这个问题就是可判定的，否则这个问题就是不可判定的。

图灵希望证明希尔伯特的判定问题是不存在一般解决方案的，即无法找出一个适用于全部情况的算法。他需要设计一个可以被证明是不可判定的判定问题。这就是本章的主题。我们必须给出一些判定问题，并且证明没有图灵机可以确定这些问题。

在本章结尾，我们会讨论机器是否会接纳自己的编码。事实上，我们可以把它表述为两个判定问题：第一个是关于有限自动机的，第二个是关于图灵机的。这两个问题分别是：

给定一个有限自动机，它会接纳自己的编码吗？

给定一个图灵机，它会接纳自己的编码吗？

先来考虑第一个问题。给定任意有限自动机，我们能不能设计一个算法，判断这个自动机是否会接纳自己的编码？很明显，答案是可以。给定任意有限自动机，我们可以找出算法来编码这个有限自动机。之后，我们可以将这个有限自动机的编码作为输入来运行这个机器。我们知道，在经历有限时间后，这个过程将会停止。如果在接纳状态停止，这个问题的答案就是“可以”，如果在其他状态停止，答案就是“不可以”。我们找到了总能给出正确答案的简单算法。因此，这个判定问题的答案就是可以确定的。

那么，第二个判定问题呢？我们似乎可以用与第一个问题类似的解决方法证明这个问题是可以确定的，但是这里存在一个问题。在论证中，我们使用了一个事实：我们在任意输入上运行有限自动机，它总会停止（步骤数等于输入字符串长度）。但是，对于图灵机来说，有三种可能结果：它可以在接纳状态停止，可以在拒绝状态停止，或者永远不停止。从我们已经掌握的知识来看，我们不清楚第三种可能性——永不停止，是否是必须的。“永远不会停止的图灵机可能是设计的问题，如果改善设计，我们就能得到永远会停止的图灵机”，这种说法似乎有一定道理，但事实并非如此。对于这些总能停止的图灵机来说，存在一些无法全部正确解答的问题。现在，我们暂时不考虑第二个判定问题，在后文中证明它是不可判定的。

除了第二个判定问题，我们还会证明不可能设计出一个算法，告诉我们一个计算机程序是会停止还是永远运行下去——这就是所谓的停机问题。

我们给出的证明使用了两个关键理念。一个理念是通过矛盾法来证明：假设某个陈述的否定，然后推导出矛盾，以此证明原命题成立。另一个理念是源自格奥尔格·康托尔的对角论证法，我们将通过介绍伯特兰·罗素提出的悖论来给出这个方法。因为许多读者可能没有见过使用矛盾法的证明，我们首先会解释这个方法如何使用，以证明“2的平方根不是有理数”为例。

矛盾证明法

在所有证明过程中，我们都试图证明一些陈述是正确的。让我们用P表示我们将要证明的陈述。存在两种基本类型的证明：直接证明和间接证明。在直接证明中，我们提出我们认为正确的陈述，并且给出符合逻辑的论证，其中每一步论证都由之前的论证推理得到。论证最后会得出你想证明的命题P的结论。在矛盾证明法中，我们采用不同的方法来论证。

我们想要证明P是正确的，但是我们先假设P是错误的，并以此开始论证。我们会像上面提到的那样一步接一步进行论证。最后我们推导出一个矛盾，即某些证明既是正确的，又是错误的。因为我们假定数学是连续的，并且没有陈述是既对又错的，所以错误出现在证明过程。我们的论证是严谨的，每一步都是由上一步推导得出的，所以唯一可能出错的地方就是假设本身，即我们假设P是错误的。因此，P一定是正确的。

矛盾证明法似乎有些复杂，但它却是数学中一个常用的强大工具。[注]我们的目标是证明没有算法能够确定第二个判定问题，也就是说我们必须证明这个算法不存在。矛盾证明法常用于这种对不存在问题的证明中。为了详细说明，我们将给出一个重要例子，即证明2的平方根不是有理数。我们先回顾一下整数和有理数的定义。

整数由0和正整数、负整数组成。整数常用的标记是，这个标记源自德语单词zahlen（计数）。我们可以写出：

$$\mathbb{Z} = \{\cdots-2, -1, 0, 1, 2, \cdots\}$$

有理数是分子为整数、分母为正整数的分数。有理数的标记是 $\mathbb{Q}$ ，源自

quotient（商）。注我们可以写出：

$$\mathbb{Q} = \{m/n: \text{其中 } m, n \text{ 属于} \mathbb{Z}\text{，且} n > 0\}$$ 注

给定一个有理数m/n，我们总能去除分子和分母的公因子，得到这个有理数的最简形式。例如，有理数17/34化简后得到1/2。当分数是最简形式时，就不可能找出分子和分母的公因子。

“如果一个整数的平方是偶数，那么这个整数也是偶数”，我们也会在证明中利用这点。现在，我们来证明2的平方根不是有理数，那么陈述就是$\sqrt{2}$不是有理数。我们将使用矛盾法来证明。我们首先假设P是错误的，然后推理出一系列结果，最终产生矛盾。比如，“$\sqrt{2}$不是有理数”是错误的，等于说$\sqrt{2}$是有理数。

证明过程如下：

1. 首先假设$\sqrt{2}$是有理数。

2. 这意味着 $\sqrt{2} = m/n$ ，其中m和n是正整数。

3. 整数m和n或许有公因子，但是我们总能化简到最简形式。所以，我们可以写出 $\sqrt{2} = p/q$ ，其中p和q是正整数，并且p和q没有公因子。

4. 对等式两边同时平方，$(\sqrt{2})^2 = (p/q)^2$，即 $2 = (p/q)^2$。。

5. 等式两边同时乘q^2，得到$2q^2 = p^2$。

6. 很明显，$2q^2$是偶数，因此，p^2也一定是偶数。

7. 因为p^2是偶数，所以p一定是偶数。

8. 因为p可以被2整除，所以我们可以写出$p = 2k$，即k是p除以2后得到的整数。

9. 将$p = 2k$带入步骤5，得到$2q^2 = (2k)^2$。

10. 因此，$2q^2 = 4k^2$。

11. 约去2，得到$q^2 = 2k^2$。

12. 因为，$2k^2$是偶数，q^2一定是偶数，所以q一定是偶数。

13. 步骤12和步骤7证明，p和q都是偶数，这意味着它们有公因子2。

14. 步骤3说明，p和q没有公因子。

15. “p和q有公因子”与“p和q没有公因子”成为矛盾。

16. 因为我们推导出了一个矛盾，所以证明一定存在错误。唯一可能出错的地方是证明开始的部分——我们假设$\sqrt{2}$是有理数。所以这个假设是错误的，$\sqrt{2}$不是有理数。

人们通常会说这个证明确定了$\sqrt{2}$是无理数，但事实并非如此。它证明了$\sqrt{2}$不是有理数。为了证明$\sqrt{2}$是无理数，你必须证明$\sqrt{2}$是实数。一旦你知道$\sqrt{2}$是实数，但不是有理数，就可以推导出$\sqrt{2}$是无理数。

罗素的理发师

罗素使用康托尔的对角论证法证明了集合的原始定义是存在缺陷的。我们将探讨罗素和康托尔都做了哪些研究，罗素提出的一个悖论可以帮助我们用一种简单的方法解释这个论证的关键。我们可以用矛盾证明法来描述罗素的“理发师悖论”：

> 假设有一个需要定期刮胡子的男士和一个理发师。有一个小镇，镇上所有男士都会定期给自己刮胡子，或者定期让理发师给他们刮胡子。假设我们得知，理发师会给镇上所有不亲自刮胡子的男士刮胡子，而且理发师不会给镇上所有亲自刮胡子的男士刮胡子。

仔细阅读上面这段文字，思考一下关于理发师住在哪里，我们能推导出什么。

我们将证明他并不住在镇上。我们再次使用矛盾证明法。

证明：我们假设理发师住在镇上。

这里有两种可能性。理发师给自己刮胡子，或理发师不给自己刮胡子。我们会依次考虑。

如果理发师给自己刮胡子，那么他就是住在镇上并且给自己刮胡子的男士，但是我们已知理发师不会给镇上自己刮胡子的男士刮胡子，这是一个矛盾。

如果理发师不给自己刮胡子，那么他是住在镇上且不自己刮胡子的男士，但是我们已知理发师会给镇上不自己刮胡子的男士刮胡子。这又是一个矛盾。

我们最初假设理发师住在镇上，证明过程告诉我们，这个假设会导致矛盾。这意味着这个假设是错误的，所以我们证明了理发师并不住在镇上。

现在，让我们回到对问题的描述，并且增加一句“理发师住在镇上”。现在我们得到了：

> 假设有一个需要定期刮胡子的男士和一个理发师。有一个小镇，镇上所有男士都会定期给自己刮胡子，或者定期让理发师给他们刮胡子。假设得知，理发师会给镇上所有不亲自刮胡子的男士刮胡子，而且理发师不会给镇上所有亲自刮胡子的男士刮胡子。另外，假设理发师住在镇上。

我们遇到了问题。前面的论证显示，理发师不能住在镇上，而我们增加了一个矛盾陈述：理发师住在镇上。现在我们得到了一个真正的悖论。这就是理发师悖论的一般形式。但是这不是一个有用的形式，或者说这不是我们需要的形式。我们对这个问题进行一些调整。

符合下列所有描述的男士是否存在？

1．他是一个需要定期刮胡子的男士，并且是一个理发师。

2．存在一个镇子，镇上居住的所有男士要么定期自己刮胡子，要么定期找理发师刮胡子。

3．理发师会给镇上所有不亲自刮胡子的男士刮胡子，而且理发师不会给镇上所有亲自刮胡子的男士刮胡子。

4．理发师住在镇上。

这个问题的答案很简单——没有。这样的理发师并不存在。

如果我们没有指定理发师是否住在镇上，这个理发师就存在，并且住在镇外。如果我们坚持认为这个理发师住在镇上，那么我们只能得到一个结论：这个理发师并不存在。

不接纳自己的编码的有限自动机

在上一章结尾，我们讨论了有限自动机M_9，并且证明它会拒绝自己的编码〈M_9〉。我们同样证明了M_2会拒绝自己的编码〈M_2〉，但是M_3会接纳自己的编码〈M_3〉。这告诉我们，存在两类有限自动机：一类可以接纳自己的编码，另一类不可以接纳自己的编码。我们现在要问，是否存在一种机器M_{FA}，可以告诉我们哪种有限自动机不会接纳自己的编码？如果存在，机器M_{FA}将只接纳拒绝自己的编码的有限自动机。因此，M_{FA}将接纳〈M_9〉和〈M_2〉，但不接纳〈M_3〉。我们将证明M_{FA}不能是有限自动机。

证明：我们假设M_{FA}是一个有限自动机。

这里有两种可能性。M_{FA}接纳〈M_{FA}〉，或拒绝〈M_{FA}〉。我们会依次证明。

如果M_{FA}接纳〈M_{FA}〉，那么它是一个接纳自己的编码的有限自动机，但是根据定义，M_{FA}只接纳拒绝自己的编码的有限自动机的编码。这是一个矛盾。

如果M_{FA}不接纳〈M_{FA}〉，那么它是一个不接纳自己的编码的有限自动机，但是根据定义，M_{FA}接纳拒绝自己的编码的有限自动机的编码。这又是一个矛盾。

我们开始的假设是M_{FA}是一个有限自动机，这个假设导致了矛盾。这意味着这个假设是错误的，所以我们证明了M_{FA}不能是有限自动机。

在上一节，我们证明了理发师不能住在镇上，但是这导致了一个问题：他住在其他地方吗？我们也可以提出类似的问题。M_{FA}是图灵机吗？答案当然是肯定的。我们已经证明，存在一种算法来判断某个有限自动机是否接纳自己的编码。因此，存在图灵机可以完成这个任务。

不接纳自己的编码的图灵机

现在，我们将注意力重新放到图灵机上，讨论图灵机是否会接纳自己的编码。就像有限自动机一样，存在可以接纳自己的编码的图灵机，同样存在不接纳自己的编码的图灵机。图灵机和有限自动机的差异在于，如果一个有限自动机不接纳自己的编码，那么它就会拒绝自己的编码，但是对图灵机而言，存在第三种可能的结果。已知图灵机拒绝自己的编码并不等于图灵机会拒绝自己的编码。图灵机可能永远不会在自己的编码上停止。事实证明，这种可能性，即图灵机发生了偏离，在讨论图灵机能做什么、不能做什么的时候相当重要。

我们依然可以借助罗素的理发师悖论来证明不存在“只接纳拒绝自己的编码的图灵机的编码”的图灵机。

证明：我们首先假设图灵机〈M_{TM}存在。

存在两种可能。〈M_{TM}接纳〈M_{TM}〉，或拒绝〈〈M_{TM}〉。我们将依次考虑两种情况。

如果〈M_{TM}接纳〈〈M_{TM}〉，那么这是一个接纳自己的编码的图灵机，但是根据定义，〈M_{TM}只接纳拒绝自己的编码的图灵机的编码。这是一个矛盾。

如果〈M_{TM}不接纳〈〈M_{TM}〉，那么这是一个拒绝自己的编码的图灵机，但是根据定义，〈M_{TM}只接纳拒绝自己的编码的图灵机的编码。这又是一个矛盾。

我们假设〈M_{TM}存在，并且证明了这会导致矛盾。这意味着我们的假设是错误的，我们证明了M_{TM}不存在。

这个证明告诉我们，不存在能够告诉我们“一个图灵机是否会拒绝自己的编码”的算法。这就意味着，“给定一个图灵机，它是否会接纳自己的编码”这个问题是不可判定的。在这个问题中加入否定会令问题变得相当复杂，但是存在另外一种说法让它看上去简单一些。

假设我们有一个可判定的判定问题P，就有可能设计出一个总会停止的图灵机。如果答案是“可以”，它会在接纳状态停止，如果答案是“不可以”，它会在拒绝状态停止。如果我们交换这个图灵机的接纳状态和拒绝状态，就得到了可以确定P的否定的图灵机。这个论述告诉我们，如果一个判定问题是可判定的，那么它的否定也是可确定的。这个论述同样告诉我们，如果一个判定问题是不可判定的，那么它的否定一定是不可判定的。[注]

因此，我们证明了判定问题“给定一个图灵机，它是否会接纳自己的编码”是不可判定的。

我们发现了一种不可判定的问题，并且已经证明它是不可判定的。这个发现十分重要，因为它证明了存在不可判定的判定问题。希尔伯特坚信，并不存在不可判定的问题，而我们已经证明了他是错的。

我们的判定问题看起来相当复杂，与我们真正想要讨论的计算机和编程问题没有太大的关系。幸运的是，我们并不需要太费力气，就可以找出关联度更高的不可判定问题。

“图灵机是否会在自己的编码上偏离”是不可判定的

当我们在一个输入字符串上运行图灵机时，存在三种可能的结果：它在接纳状态停止；它在拒绝状态停止；它永远不会停止。让我们考虑一下与之相关的三个问题。

是否可以设计一个接纳图灵机编码作为输入的图灵机，当且仅当被编码的图灵机接纳自己的编码时，被设计出的图灵机会在接纳状态停止？

是否可以设计一个接纳图灵机编码作为输入的图灵机，当且仅当被编码的图灵机拒绝自己的编码时，被设计出的图灵机会在接纳状态停止？

是否可以设计一个接纳图灵机编码作为输入的图灵机，当且仅当被编码的图灵机在自己的编码上发生偏离时，被设计出的图灵机会在接纳状态停止？

第一个问题的答案是“可以”。我们可以设计一个图灵机A，将〈M〉作为输入，在输入〈M〉上模拟运行图灵机M。因此，如果M在接纳状态停止，那么A接纳〈M〉。如果M在拒绝状态停止，那么A拒绝〈M〉。如果M在〈M〉上发生偏离，那么模拟永远不会停止，因此A将在〈M〉上偏离。只要M接纳〈M〉，我们的机器A就将在接纳状态停止。

第二个问题的答案也是“可以”。我们可以通过交换A的接纳状态和拒绝状态设计一个图灵机B。这意味着B将〈M〉作为输入，在M上模拟运行〈M〉。如果M在接纳状态停止，那么B拒绝〈M〉。如果M在拒绝状态停止，那么B接纳〈M〉。如果M在〈M〉上偏离，那么模拟同样永远不会停止，B将在〈M〉上偏离。所以只要M拒绝〈M〉，我们的机器B就将在接纳状态停止。

第三个问题的答案是“不可以”。这里，我们将继续使用矛盾证明法证明这一点。

证明：我们假设存在图灵机C，接纳图灵机的编码作为输入，当且仅当被编码图灵机在自己的编码上偏离时，这个被设计出的图灵机在接纳状态停止。

我们将描述一个新的图灵机D，这个图灵机同时包括B和C。给定任意图灵机的编码〈M〉，在B和C上同时运行〈M〉。如果B或C接纳〈M〉，那么D接纳〈M〉。如果B和C都不接纳〈M〉，至少一个拒绝〈M〉，那么D拒绝〈M〉。如果B和C都在〈M〉上偏离，那么D也在〈M〉上偏离。

机器D接纳图灵机编码作为输入。当且仅当B或C在接纳状态停止，D在接纳状态停止。如果被编码的图灵机拒绝输入，或在输入上偏离，这种情况就会发生。这意味着，当且仅当被编码的机器不接纳自己的编码时，D才会在接纳状态停止。这就产生了矛盾，因为我们已经证明不存在具备这种性质的图灵机。

结论是不存在接纳图灵机编码作为输入的图灵机，当且仅当被编码的图灵机在自己的编码上偏离时，被设计出的图灵机将在接纳状态停止。

这证明了问题“图灵机是否会在自己的编码上偏离”是不可判定的，“图灵机是否会在自己的编码上停止”这个问题也是不可判定的。

接纳、停机和空白磁带问题

我们现在得到了两个不可判定的问题：“图灵机是否会接纳自己的编码”和“图灵机是否会在自己的编码上停止”我们将概括这两个问题。

接纳问题：给定任意图灵机M和任意输入I，M会接纳I吗？

停机问题：给定任意图灵机M和任意输入I，M会在I上停止吗？

我们再考虑一下停机问题的变体。

空白磁带问题：给定任意图灵机M和一个空白输入磁带，M会停止吗？

我们先来看看接纳问题。我们能否设计一个图灵机，接纳任意图灵机的编码〈M〉和任意输入I，当M接纳I时在接纳状态停止，当M不接纳I时在拒绝状态停止？答案当然是“不可以”，因为我们无法设计出一个仅在输入I为〈M〉时成立的图灵机。如果我们无法设计出回答简单问题的图灵机，就无法设计出回答复杂问题的图灵机，将简单问题作为特例。因此，接纳问题是不可判定的。

停机问题就像在问一个给定图灵机是否会在自己的编码上停止，因此停机问题也是不可判定的。

我们重新表述停机问题：是否存在一个算法，将程序和数据作为输入，并且告诉我们程序是否会在数据上停止？正如我们已经证明的，这个问题是不可判定的。如果计算机程序员能够得到一个告诉他有些程序永远不会在某个输入上停止的工具，会非常有帮助，然而并不存在这样的工具。

但是，如果我们限定程序和数据的种类，这个问题就会变成可以确定的。我们或许能找到一个不住在镇上的理发师。

我们要考虑的最后一个问题是空白磁带问题。这个问题像是停机问题的特例——输入磁带一直是空白的，但是这里有一个小技巧，如果你能够找出空白磁带问题的决策过程，那么你就拥有了停机问题的决策过程。因为停机问题是不可判定的，这样的决策过程并不存在。

给定任意图灵机M和任意输入I，你能够设计一个新的图灵机M_I。当给出一个空白磁带时，这个新图灵机首先在磁带上写下I，随后在输入I上模拟运行M。M_I将在空白磁带上停止，当且仅当机器M在I上停止。

一个不可计算函数

图灵机拥有用于输入和输出的线性磁带。如果我们限制该机器只有一个磁带，磁头就会在一次计算中来回活动。匈牙利数学家蒂博尔·劳多（Tibor Radó）决定研究磁头在进入接纳状态前最多会来回移动多少次。

1962年，劳多基于这一理念发明了忙碌的海狸函数。实际上，他根据同一理念发明了几个略微不同的函数。其中最容易研究的就是最大移位函数，用$S(n)$来表示。

我们将研究只被允许写下两个符号的图灵机。它们通常由1和0来表示，其中0代表空白符号。机器总是从空白磁带开始。我们想知道，图灵机在停止前完成的步骤数如何与图灵机的状态数产生联系。

我们将用n来表示图灵机的状态数（根据劳多的说法，接纳状态和拒绝状态没有被计入这个数值）。考虑到所有拥有n个状态的图灵机，其中一些会在空白磁带上停止，一些则不会。当然，在那些会停止的图灵机中，一定至少存在一个图灵机会执行最多的步骤数，直到最后停止。我们用$S(n)$来表示这个最多步骤数。

在n值一定的情况下，为了计算这个函数，你需要设计出所有含有n个状态的图灵机，然后在空白磁带上运行它们。你需要计算步骤数，直到它们停止。最大步骤数即为$S(n)$的值。这个过程的问题在于一些机器需要相当长的时间才能停止，一些机器则永远不会停止。如果有一个机器已经运行了很长时间，并且尚未停止，你如何区分它属于哪个种类？尽管存在这个问题，我们还是知道一些$S(n)$的值。

$$S(1) = 1, S(2) = 6, S(3) = 21, S(4) = 107$$

令人惊讶的是，这些是全部的已知值。[注]我们还知道$S(5)$ H 47 176 870和$S(6) > 7.4 \times 10^{36\,534}$。正如你所见，即便我们不知道多少准确数值，这些数值也在快速增长。实际上，我们可以证明$S(n)$不是一个可计算函

数。注

如果我们能够找出一个图灵机，当我们输入任意自然数n的时候输出$f(n)$，那么$f(n)$就是一个可计算函数。如果存在一个算法能够计算给定n的$f(n)$，我们就说$f(n)$是一个可计算函数。

如果$S(n)$是可计算函数，我们就能够给出一个过程去判断下面提到的空白磁带问题：给定任意拥有一个空白磁带的图灵机，首先计算它的状态数，随后使用算法计算它$S(n)$的值。之后运行M。它可能会在一定步骤后停止，但是如果在$S(n)$次步骤后没有停止，你就知道它永远不会停止。因此，如果$S(n)$是一个可计算函数，那么空白磁带问题就是可判定的。我们已经证明空白磁带问题是不可判定的，$S(n)$就一定不是可计算函数。

图灵的方法

停机问题是关于任意程序和输入数据的判定问题。这个问题关注程序是否会在数据上停止。实际上，这个问题是不可判定的，没有算法能够在所有情况下正确判定这个问题。停机问题或许是最著名的不可判定的判定问题。然而，图灵并未在自己的论文中介绍这个问题。

图灵机并没有接纳状态。它们被设计用来计算实数，因此如果计算一个无理数，它就永远不会停止。斯蒂芬·克莱尼和马丁·戴维斯调整了图灵机的概念，以包含接纳状态。一旦你拥有这种新的图灵机，就能考虑停机问题。正是戴维斯提出了停机问题这一概念。注

如果你已经理解了“程序”和“输入”这类术语，并且理解了程序在输入上停止意味着什么，停机问题就很容易理解。因为如今无处不在的计算机，我们对这些概念并不陌生，但是在1936年，人们还没有发明计算机。对于从未听说过计算机的人来说，停机问题很难描述。图灵并不想使用这样一个有关假设中的计算设备性质的不可判定问题。他希望使用的不可判定问题是数学家能够立即理解的问题。图灵借助乔治·康托尔的观点，发现了一个这样的问题。

---

1. 希腊古典哲学家用他们自己的论证使用这项技术。本书中，它通常指代reductio ad absurdum。

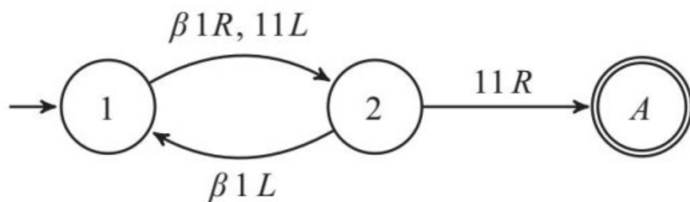
2. 它实际上源自quoziente，quotient（商）的意大利语单词。意大利数学家皮亚诺发明了这个词汇，并于1895年首先使用了这一标记。

3. 这个定义并不是很正确。我们想要1/2，2/4和17/34表示同一个数字。

我们目前定义有理数的方式让它们看起来是不一样的数。这不应该导致任何麻烦，但是严格来讲，我们应该定义一个等价关系，之后定义有理数是等价类——1/2, 2/4和17/34都属于同一个等价类。

4. 另一种描述不确定性的方法是与语言相关的图灵机。如果图灵机的补集仍然是图灵机，那么这个语言是可判定的。可判定性语言对应可判定性问题。我们在有限自动机那一章提到过，一个正则语言的补集仍然是正则语言，这说明所有正则语言都是可判定的。可判定语言通常也被称作“递归的”。全集上的图灵机语言被称作“递归可枚举的”。递归可枚举的语言包含递归语言。

5. 我们知道，$S(2) = 6$。图注–2是一个机器，给定一个空白输入磁带，在停止前执行了6步。



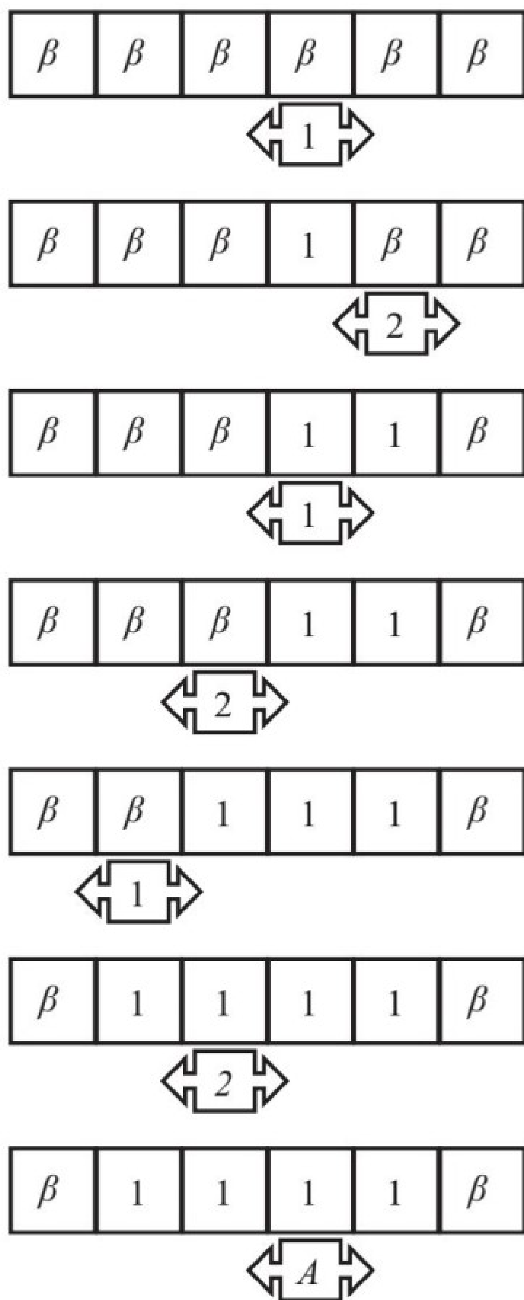
图注–2 执行了6步的机器

图注–3是计算过程。

6. 斯科特·阿伦森的网页中有他的多篇文章。其中之一是《谁能命名更大的数字》。这篇文章与$S(n)$有关，非常值得一读。

7. 马丁·戴维斯是波斯特的本科生，邱奇的研究生，香农的助理。他因为对希尔伯特的第10个问题的研究赢得过不少奖项，对证明“该问题是不可判定的”贡献了重要力量。

他同样是一位伟大的评论家。他的《可计算性和不可解性》于1958年出版，最早探讨了有关图灵机的计算原理。这本书以一种便于计算机科学家理解的方式，呈现了只出现在逻辑学领域的想法。



图注–3 计算过程

第八章 康托尔的对角论证法

我能看到它，但我并不相信它！

——格奥尔格·康托尔

我并不清楚康托尔的理论中什么占据了主导地位——哲学还是神学，但是我确定其中没有数学。

——利奥波德·克罗内克（Leopold Kronecker）

没有人可以让我们离开康托尔创造的天堂。

——戴维·希尔伯特

11岁那年，格奥尔格·康托尔随家人从圣彼得堡迁往德国。1867年，在柏林大学完成论文后，康托尔进入哈勒大学继续自己的学术研究。

康托尔对集合的“大小”（size）的概念很感兴趣，特别是无限集合。他的想法独树一帜，并且带来了一些令人吃惊的结果，比如有无数个更大的无穷大。与弗雷格相似，康托尔也采用了集合的直观方法。他意识到，所有可能集合的集合给他的理论带来了问题，问题在于是否只需要进行一些小的修改就能够带来改善。

从本章开头引用的几句话中，我们能够看到当时学界对康托尔研究成果的反应。在一些数学家眼中，他的研究就是胡诌。利奥波德·克罗内克、亨利·庞加莱（Henri Poincaré）都对他进行了抨击。在宗教领域，康托尔也没能幸免。如果上帝就是无穷，那么有不同的无穷是否意味着有无穷多个上帝？以希尔伯特为代表的另外一群数学家却开始支持康托尔，在他们看来，康托尔不仅是正确的，而且对数学的未来至关重要。

在第一章中，我们讨论了数学基础。这其中既包括希尔伯特的形式主义方法，也包括罗素和怀特海的逻辑学方法。逻辑学派希望证明，数学完全来源于逻辑。形式主义派则希望证明存在一个形式系统，能够让你构建有关公理的一致性和完备性的理论。这两种方法都假设数学并不存在根本错误。

不过，另一群数学家的观点却截然不同。在他们看来，康托尔的研究大错特错，他们甚至认为有必要改写数学的基础，以此排除康托尔采用的论证

方式。这就是1908年左右由德国数学家L·E·J.布劳威尔（L. E. J Brouwer）建立的直觉论派。

康托尔可能患有双相情感障碍，不少抨击都针对他本人，这也导致在坎坷的一生中，他曾多次住院。克罗内克的反应让他陷入了极度的悲伤。康托尔一直对自己的恩师克罗内克毕恭毕敬。然而克罗内克却认为康托尔的理论大错特错，简直算得上是数学界的败类。克罗内克对康托尔本人及其研究的批评，从来都是公开且严厉的。

如今，大多数数学家都是康托尔的支持者，他们中的很多人都认同希尔伯特对康托尔的评价："没有人可以让我们离开康托尔创造的天堂。"

基数

康托尔想知道，两个集合大小相同到底意味着什么。对有限集合来说，回答这个问题只需数出集合中的元素数。{a, b, c}和{1, 2, 3}拥有相同的大小，因为它们都有三个元素，但是康托尔想把这个理念拓展到无限集合。他决定使用双映射的概念。

一个双映射是指，在集合A和B之间，集合A的每个元素都与集合B的每个元素按照某种方式一一对应：A中的每个元素都与B中的一个元素相对应，而B中的每个元素也与A中的一个元素相对应，每个集合中的一个元素，都与另一个集合中的唯一元素相对应。如果能够在两个集合之间找出一个双映射，康托尔就定义两个集合拥有同样的基数。如果无法在两个集合之间找出一个双映射，那么它们就没有相同的基数。

我们将使用绝对值标记来表示一个集合的基数。因此，$|A|$表示A的基数。如果集合A和B拥有相同的基数，我们就会写成$|A| = |B|$。

我们使用函数标记来描述双映射。对于例子$A = \{a, b, c\}$ 和$B = \{1, 2, 3\}$，我们可以定义一个函数$f: A \rightarrow B$，其中$f(a) = 1$，$f(b) = 2$，$f(c) = 3$。A和B中的每个元素都在另一个集合中有唯一的对应元素。因此，函数f描述了一个双映射。我们已经在A和B之间显式给出了一个双映射，所以我们证明了它们拥有相同的基数，可以写成$|A| = |B|$。对于有限集合，这似乎不是什么重要的结论。有限集合的基数等于该集合中的元素个数。在我们的例子中，可以写出$|A| = |B| = 3$。

当我们把注意力转移到无限集合上时，事情会变得更加有趣。我们知道，自然数严格包含偶数注；整数严格包含自然数；有理数严格包含整数；实数严格包含有理数。可以用下面的表达式表示：

$$2\mathbb{N} \subset \mathbb{N} \subset \mathbb{Z} \subset \mathbb{Q} \subset \mathbb{R}$$

如果我们向一个集合中增加元素，基数不会减少。因此，我们得到了：

$$|2\mathbb{N}| \leqslant |\mathbb{N}| \leqslant |\mathbb{Z}| \leqslant |\mathbb{Q}| \leqslant |\mathbb{R}|$$

对于有限集合来说，如果你想加入更多元素，可以增加集合中的元素数量，而基数一定会增加。令人吃惊的是，对于无限集合来说，这一点未必正确。我们将证明

$$|2\mathbb{N}| = |\mathbb{N}| = |\mathbb{Z}| = |\mathbb{Q}| \leqslant |\mathbb{R}|$$

有理数的子集拥有相同的基数

为了证明两个集合拥有相同的基数，我们必须证明它们之间存在双映射。首先，我们想证明$|2\mathbb{N}| = |\mathbb{N}|$。

为了证明这点，我们必须设计一个双映射$f: \mathbb{N} \to 2\mathbb{N}$，即集合$\{1, 2, 3, 4, \ldots\}$中的元素与集合$\{2, 4, 6, 8, \ldots\}$中的元素一一对应。这一点相当容易实现。假设函数$f$的定义是$f(n) = 2n$。这会将$\mathbb{N}$中的数字$n$与$2\mathbb{N}$中的偶数$2n$配对。给定$\mathbb{N}$中的一个数字，让其乘以2，就能找到它在$2\mathbb{N}$中的对应数字。给定$2\mathbb{N}$中的任意数字，让其除以2，就能找到它在$\mathbb{N}$中的对应数字。很明显，这是一个双映射，我们证明了$|2\mathbb{N}| = |\mathbb{N}|$。

你第一次看到这个等式时，可能会有些吃惊。很明显，$\mathbb{N}$包含的元素要多于$2\mathbb{N}$。它不仅包含偶数，还包含奇数。看起来，$\mathbb{N}$的基数应该是$2\mathbb{N}$的2倍。实际上，这一点是正确的，但是我们将看到，让一个无限数乘以2，并不会得到一个更大的基数。让一个无限基数乘以2，会得到相同的无限基数。

无限集合的这个性质非常令人费解，并不适用于有限集合。给定一个无限集合A，总能找出一个不包含A中全部元素的子集B，但是B的基数与A的基数完全相同。这意味着，A中的每一个元素都与B中的一个不同元素相对

应。这个结论与我们的直觉相去甚远，但是之所以有这个结论，是因为我们的直觉建立在不会出现的有限集合之上。现在这个性质已经被数学界接受，但是很容易想见当时为什么康托尔的理念没有被立即接受。

康托尔命名了自然数的基数。他用$\aleph_0$来表示这个数字。(注)自然数和偶数之间的双映射说明了$|2\mathbb{N}|=|\mathbb{N}|=\aleph_0$。

类似的论证可以被用于证明奇数的基数同样是$\aleph_0$。

如果A和B是不相交的集合，即两个集合不包含任何相同的元素，那么$|A\cup B|=|A|+|B|$。在我们的例子中，$A=\{a, b, c\}$且$B=\{1, 2, 3\}$，我们得到了$A\cup B=\{a, b, c, 1, 2, 3\}$。请注意，$|A\cup B|=6=3+3=|A|+|B|$。对于有限集合来说，这一点非常直接，但是现在让我们回到无限集合。让A等于偶数，B等于奇数。集合A和B是不相交的。它们的并集是自然数N。我们知道$|A\cup B|=|A|+|B|=$，这等于在说$\aleph_0=\aleph_0+\aleph_0$。这个看起来奇怪的论述，其实是正确的。

这告诉我们，在处理基数的时候要非常小心。它们不同于普通数字，基数加法会按照我们描述的方式进行，尽管它总会产生一些奇怪的陈述，比如$\aleph_0=\aleph_0+\aleph_0=\aleph_0+\aleph_0+\aleph_0=\aleph_0+\aleph_0+\aleph_0+\aleph_0$，但这并不会导致任何矛盾。

我们已经证明了$|\mathbb{N}|=|2\mathbb{N}|=\aleph_0$。现在，我们将证明所有整数的基数同样是$\aleph_0$。整数包含正整数、负整数和0。自然数（正整数）的基数为$\aleph_0$。自然数的否定（负整数）的基数也是$\aleph_0$。只包含0的集合的基数为1。这意味着整数的基数$|\mathbb{Z}|$是$\aleph_0+\aleph_0+1$。很明显，$\aleph_0+\aleph_0\leqslant\aleph_0+\aleph_0+1\leqslant\aleph_0+\aleph_0+\aleph_0$。

我们知道，$\aleph_0+\aleph_0$和$\aleph_0+\aleph_0+\aleph_0$都等于$\aleph_0$，因此我们可以用$\aleph_0$代替不等式两边，得到$\aleph_0\leqslant\aleph_0+\aleph_0+1\leqslant\aleph_0$。

这证明(注)，$\aleph_0+\aleph_0+1$同样等于$\aleph_0$，我们推导出$|\mathbb{Z}|=\aleph_0$。现在，我们得到了$|2\mathbb{N}|=|\mathbb{N}|=|\mathbb{Z}|=\aleph_0$。

下一步就是证明有理数的基数同样是$\aleph_0$。我们先来看看正有理数$\mathbb{Q}^+$。因为正有理数包含自然数，我们可以推导出它们的基数大于或等于$\aleph_0$，即|

$\mathbb{Q}^+| \geq \aleph_0$。

现在，我们将证明 $|\mathbb{Q}^+| \leqslant \aleph_0$。我们只关注 $p/q \in \mathbb{Q}^+$，其中 p 和 q 没有公因子。我们可以定义一个函数 $f: \mathbb{Q}^+ \to \mathbb{N}$，其中 $f(m/n) = 2^m 3^n$。对我们的例子来说，$f(1/2) = 2^1 \times 3^2 = 18$。

这个函数当然不是一个双映射，因为 $5 \in \mathbb{N}$ 在 $\mathbb{Q}^+$ 中并没有对应的数字。但是，请注意在 $\mathbb{Q}^+$ 中的每个元素都在中有对应数字，而且在 $\mathbb{Q}^+$ 中没有哪两个元素在 $\mathbb{N}$ 中有同一个对应元素。因此，我们发现了 $\mathbb{Q}^+$ 和 $\mathbb{N}$ 的某个子集之间的双映射，也就是说 $|\mathbb{Q}^+| \leq \aleph_0$。

我们已经证明了 $\aleph_0 \leqslant |\mathbb{Q}^+| \leqslant \aleph_0$。通过这一点，我们可以得到 $|\mathbb{Q}^+| = \aleph_0$。因此，我们知道 $|2\mathbb{N}| = |\mathbb{N}| = |\mathbb{Z}| = |\mathbb{Q}^+| = \aleph_0$。

我们同样证明了 $\aleph_0 + \aleph_0 + \cdots + \aleph_0 = n\aleph_0 = \aleph_0$。

有理数的形式为 p/q。分子和分母都拥有 $\aleph_0$ 种选择。因此，有理数的数量是 $\aleph_0 \times \aleph_0$。这意味着 $\aleph_0^2 = \aleph_0$。

我们可以一直乘以 $\aleph_0$，以此推导出对任意 n，$\aleph_0^n = \aleph_0$。

看起来，我们只是用一个符号取代了∞，找到了一种复杂方法来表示显而易见的事情，比如两个无限集合的并集是另一个无限集合。如果我们无法证明存在更大的集合，这个说法就是正确的，但是康托尔证明了存在无限个更大的无限数。在我们开始研究这一点前，我们先来介绍“希尔伯特旅馆”。这个例子源于乔治·伽莫夫（George Gamow），并且出现在他的著作《一，二，三，……无限：科学事实和猜测》（*One, Two, Three,… Infinity: Facts and Speculations of Science*）中。这个例子的版本很多。有的时候，它被表述为悖论，但实际上，它不是悖论。它只是康托尔对基数加法研究的一个版本。

希尔伯特旅馆

希尔伯特旅馆是一个拥有无限房间的旅馆。这些房间被标号为1，2，3，以此类推。在周一晚上，这家旅馆客满。每个房间都有人居住。突然，经理接到一位客人的订房电话，这位客人希望今天晚上在这家旅馆预订一个

房间。经理向客人解释，旅馆所有房间都被预订了。而这时，实习生却说每个人（包括新到的客人）都可以有一个房间。他告诉所有住在旅馆里的人，搬到临近的拥有最大房间号的房间。如果你住在102室，现在就必须搬到103室。每个现在有房间的人都会拥有一个新房间，而1室就空出来留给新到的客人。

尽管这看起来很奇怪，但却没有什么矛盾。它确实证明了$\aleph_0+1=\aleph_0$。

所有周一晚上住在这家旅馆的人，周二也会继续住在该旅馆。周二突然来了10位客人，并且都需要房间。我们已经知道如何为所有人安排房间——每个人搬到比当前房间号大10的房间中。周一晚上从102室搬到103室的客人，现在需要搬到113室。这样就腾出了1到10的房间给新来的客人。

这一论证可以拓展到任意有限数，证明对任意n，$\aleph_0+1=\aleph_0$。

我们只描述了希尔伯特旅馆中的一种情况。在小镇的另一边还有另外一家旅馆。这家旅馆同样有无限个房间，而且每一个房间都住有客人。周三时这栋建筑的巡查员发现旅馆的地基出现了问题，并且宣称这家旅馆不适合居住。居住在那里的无限数量的客人需要新地方。这家违建旅馆的经理向第一家旅馆的经理求助，询问他们是否有无限个空闲房间。经理求助于实习生，实习生再次想出了办法。第一家旅馆的客人从当前的房间搬到两倍于自己当前房间号的房间中。那个从102室搬到103室，又搬到113室的倒霉客人，现在搬到了226室。每个住在第一家旅馆里的客人，都搬到了一个新房间。现在，这家旅馆空出了房间，留给违建旅馆的客人。实习生告诉那些原来住在违建旅馆的客人，将他们的旧房间号乘以2，减1，就是他们在第一家旅馆里的新房间号。原本在第二家旅馆住102室的客人，搬到了第一家旅馆的203室。按照这种方法，每个人都得到了一个房间。

这个论证说明了$\aleph_0+\aleph_0=\aleph_0$。

定义不完善的减法

我们已经证明了$\aleph_0+1=\aleph_0$，$\aleph_0+\aleph_0=\aleph_0$。而且，很明显$\aleph_0=\aleph_0$。我们能否重新组合上面列出的三个等式，得到$1=\aleph_0-\aleph_0$，$\aleph_0=\aleph_0-\aleph_0$和$0=\aleph_0-\aleph_0$，尽管基数之间的加法有意义，但减法却不是这样。我们认为减法的定义并不完善。乘法是定义完善的——两个基数的乘积有唯一答案，而除法对于无限基数是定义不完善的。因为涉及减法和除法的问题没有唯一答案，我们将避免进行这两种运算。我们主要关注加法和乘法。注

一般对角论证

康托尔证明了，对任意自然数n，$n\aleph_0=\aleph_0$和$\aleph_0^n=\aleph_0$。他开始寻找方法构建更大的基数。

他开始考虑一个集合的所有可能子集的集合。例如，集合$S=\{0，1，2\}$的子集包括：集合本身$S=\{0，1，2\}$；有两个元素的子集$\{0，1\}$、$\{0，2\}$和$\{1，2\}$；有一个元素的子集$\{0\}$、$\{1\}$和$\{2\}$；没有元素的子集$\{\}$。一个集合S的所有可能子集的集合被称作幂集,用$P(S)$ 表示。

$$P(S)=\{\{0,1,2\},\{0,1\},\{0,2\},\{1,2\},\{0\},\{1\},\{2\},\{\}\}\text{ 。}$$

$|S|=3$，而且$|P(S)|=8=2^3=2^{|s|}$。

对于有限集合S，$|P(S)|=2^{|s|}$。之所以有这样的结论，是因为当我们从集合S构建子集时，每个元素都有两种可能，即包含它或不包含它。在我们的例子中，我们可能会用I来表示包含（include），用E来表示不包含（exclude）。字符串EIE就代表不包含第一个和第三个元素，但是包含第二个元素。在我们的例子中，EIE就代表子集$\{1\}$。使用I和E来表示所有可能的长度为3的8个字符串，分别是：III，IIE，IEI，EII，IEE，EIE，EEI，和EEE。这样一来，我们可以很清楚地理解这8个字符串如何与8个子集分别对应。

康托尔将这些概念拓展到了无限集合。我们来考虑自然数集合，并且研究N的所有子集的集合。当我们构建一个子集时，每个自然数都有两种可能。因此，我们一共有$2^{\aleph_0}$个子集。所以，$|P(N)|=2^{\aleph_0}$。

康托尔证明了一系列定理，但是下面的定理是唯一以康托尔的名字命名的定理。

康托尔定理

给定任意集合S，$|S|<|P(S)|$。

康托尔定理告诉我们$|\mathbb{N}|<|P(\mathbb{N})|$，因此$\aleph_0<2^{\aleph_0}$。我们现在得到了一个比$\aleph_0$大的无限基数。

如果我们令$A=P(\mathbb{N})$，就能研究A的所有可能子集的集合。它的基数是$2^{2^{\aleph_0}}$。康托尔定理告诉我们$|A|<|P(A)|$，也就是说$2^{\aleph_0}<2^{2^{\aleph_0}}$。按照这种方法，我们得到了一个更大的无限基数。

考虑一下所有可能集合的集合。它是最大的可能集合，所以它的基数应该是最大的可能基数。那么，所有集合的幂集的基数是多少呢？康托尔定理

证明，它还有更大的基数。但是，现在我们得到了一个悖论。我们已经发现了比最大基数还要大的基数。

根据现代观点，以这种方式定义的集合去掉了所有集合的集合。没有最大的集合，也就没有最大的基数。康托尔发现的悖论可以被当作对“没有最大基数”的证明。

现在我们来证明康托尔定理。这里的证明使用了理发师悖论的想法，但是我们应该记住，康托尔是这个论证的发明者。首先，我们借助一个例子说明这个证明背后蕴含的理念。

想象一个拥有1 000人的小镇。镇长喜欢对镇上的居民进行表格管理。她有一些只有一两个人名的表格，有一些有很多人名的表格，有一些没有任何名字的空表格。在编写完众多表格后，她意识到自己已经写了1 000张表格。因为她有1 000张表格，而且人口总数恰好也是1 000，她决定给每个人发一张表格。她开始随机向镇上的人发放表格。

镇上的人阅读了他们拿到的表格。有些人看到自己的名字出现在表格上，他们非常高兴。其他人看到自己的名字没有出现在表格上，他们很失望。镇上居民被划分为两部分：名字在表格上的开心的人和名字没有在表格上的不开心的人。

看到有一些人不开心，镇长决定为他们再做一个表格。她编写了一个表格，上面写着已经拿到了表格，但名字没有出现在那张表格上的人的名字。她开始端详这个新表格，并且好奇这张表格上列出的名字是否会与已经发出的1 000张表格中的某一张包含相同的名字。这张不开心的人的列表是一张新表格，还是原来1 000张表格中的某一张呢？

她推理如果这张表格与已经发出的1 000张表格中的某一张完全一致，那么就会有人收到这张写有不开心人的名字的表格，并且看到了这些名字。存在两种可能性。一种是如果这个人的名字出现在这张表格上，那么他就会高兴看到了自己的名字，但是这是一张写有不开心人的名字的表格，这个人的名字不应该出现在上面。因此，这种可能性不会发生。另一种是如果这个人的名字没有出现在这张表格上，那么他就会不开心，但是这意味着他的名字应该在表格上，所以第二种可能性也不会发生。

通过使用这种方法，她认识到不开心人群的表格不可能是原来1 000张表格中的一张。

我们将使用这个论证来证明康托尔定理。

证明：首先，我们假设存在一个集合S。很明显，$|S|$ G $|P(S)|$。我们必须证明这两个基数不能相等。如果我们能够证明无法构建出双映射：$f: S \to P(S)$ ，就可以证明这点。我们的目标是证明对任意集合S和任意函数$f: S \to P(S)$，f永远不可能是双映射。

给定任意$s \in S$，$f(s)$将是$P(S)$的一个元素。这意味着$f(s)$是S的一个子集。存在两种可能：s属于这个子集，或不属于这个子集。我们可以写成$s \in f(s)$ 或 $s \notin f(s)$（将s当作人，把f(s)当作这个人收到的表格）。

我们现在定义一个集合T，它包含没有出现在映像$f(s)$中的S的所有元素s。我们可以写出更正式一些的表示：$T = \{s \in S \mid s \notin f(s)\}$（这是包含了不开心的人的表格）。

集合T是S的一个子集。因此，T是$P(S)$ 的一个元素。我们现在要证明，不存在一个$t \in S$，满足$f(t) = T$。这意味着T在S中没有特别的对应，因此f不可能是双映射。

我们使用矛盾法来证明。首先假设存在$t \in S$，并且$f(t) = T$。我们必须证明这将导致矛盾。

我们可以询问t是否为T的元素。存在两种可能：是或不是。我们将证明两种可能都会导致矛盾。

如果$t \in T$，这就意味着t $f(t) = T$。这将导致矛盾，因为T被定义为包含所有不是$f(s)$元素的s。

如果$t \notin T$，这就意味着$t \notin f(t) = T$。因为T被定义为包含所有不是$f(s)$元素的s，t一定是T的元素。我们再一次推导出矛盾。

因为我们假设存在$t \in S$，$f(t) = T$导致了矛盾，所以S中一定不包含可以与T匹配的元素。因此，f不能是双映射，这也就说明了S和$P(S)$无法拥有相同的基数。

这个证明相当复杂。为了说明发生了什么，我们将研究一个具体例子：$S = \{0, 1, 2\}$ 以及 $P(S) = \{\{0, 1, 2\}, \{0, 1\}, \{0, 2\}, \{1, 2\}, \{0\}, \{1\}, \{2\}, \{\}\}$ 。我们选择 $f: S \rightarrow P(S)$，而且 $f(0) = \{0, 1, 2\}$，$f(1) = \{0\}$，$f(2) = \{1\}$。对于这个例子，$0 \in f(0)$ 但是 $1 \notin f(1)$，$2 \notin f(2)$ 。

集合T被定义为不属于$f(s)$的s。所以，$T = \{1, 2\}$。上述证明显示，不存在$t \in S$且$f(t) = T$。在我们的例子中，T不是任意S中元素的映像。

选择不同的f，证明你为每个选择得出的T，永远不是任何S中元素的映像，这是一个不错的练习。

实数的基数

在本章中，我们已经证明了 $|\mathbb{N}| = |\mathbb{Z}| = |\mathbb{Q}| = \aleph_0$，而且 $2^{\aleph_0} > \aleph_0$。现在，我们将要证明实数的基数是 $2^{\aleph_0}$。

我们开始研究介于 0 和 1 之间的实数。这些数字的标记是（0, 1）。在数学中，标记（a, b）包含了多种含义，我们经常用（0, 1）来表示 $\{x \in \mathbb{R} \mid 0 < x < 1\}$。

（0，1）之间的每一个实数x都可以被写作$0.a_1a_2a_3a_4\ldots$的十进制数。结尾处的点说明了这个十进制展开会包含无限个位置（我们无法总是以有限个位置数结束，因为如果我们这样做了，我们将得到一个有理数）。

我们在使用十进制表达的时候，有一个小小的技巧。一些实数拥有两种可能的十进制展开。例如，0.20000…与0.19999是同一个数字，其中第一个表达式中的0一直出现，而第二个表达式中的9一直出现。我们不是在说这些数字非常接近，它们实际上是相等的。这种歧义只出现在0和9当中。如果一个十进制展开在某一位后一直重复出现0，这个数字可以被改写为以无限个9为结尾的数字（只有在完备性问题中才会出现这一现象，这在后面的论证中不会导致任何问题）。

考虑$0.a_1a_2a_3a_4\ldots$形式的十进制展开。我们想知道，存在多少这样的数字。（0，1）的基数与所有按照这种形式书写的十进制数字的基数相同。对于a_1的选择，有10种可能性（数字从0到9）。对a_2，a_3等也都有10个选择。存在无限多个十进制位置，我们现在对无限数多了一些理解。这种情况下的无限是自然数的基数$\aleph_0$。因此，$0.a_1a_2a_3a_4\ldots$形式的十进制展开的数字是$10\times10\times\ldots=10^{\aleph_0}$，这意味着（0，1）的基数是$2^{\aleph_0}$。

这个表达式之所以出现 10，是因为我们在处理十进制数这个事实。我们也可以使用二进制数得到完全相同的论证。（0，1）之间的所有实数都可以写为 $0.b_1b_2b_3\cdots$。这一次，每个 b_i 有两种选择——0 或 1。我们得到的（0,1）的基数是 $2^{\aleph_0}$。含义之一是 $2^{\aleph_0}=10^{\aleph_0}$（我们可以使用任意底数得到完全相同的结论，这告诉我们，对任意 $n\geqslant2$，$2^{\aleph_0}=n^{\aleph_0}$）。

聪慧的读者可能会担心，认为（0，1）的基数是 $10^{\aleph_0}$ 或许不够准确。在我们研究所有可能的十进制展开时，出现了 $10^{\aleph_0}$ 种可能性，但是我们已经提到，有的时候两个完全不同的十进制展开对应同一个实数。存在$\aleph_0$个实数拥有两种表达式（拥有两种表达式的实数是有理数）。因此，（0，1）的基数是 $10^{\aleph_0}-\aleph_0$。与 $10^{\aleph_0}$ 相比，$\aleph_0$ 是可以忽略的，可以认为 $10^{\aleph_0}-\aleph_0=10^{\aleph_0}$。

我们已经确定，（0，1）的基数为$2^{\aleph_0}$。前面论证中的最终步骤证明，（0，1）的基数与实数集合的基数相同。我们通过在它们之间构建双映射，实现了这点。

图8–1是一个双映射的例子。图中有两条垂直渐近线$x=0$和$x=1$。随着x从右侧迫近0，y值迫近负无穷，x从左侧迫近1，y值迫近正无穷。

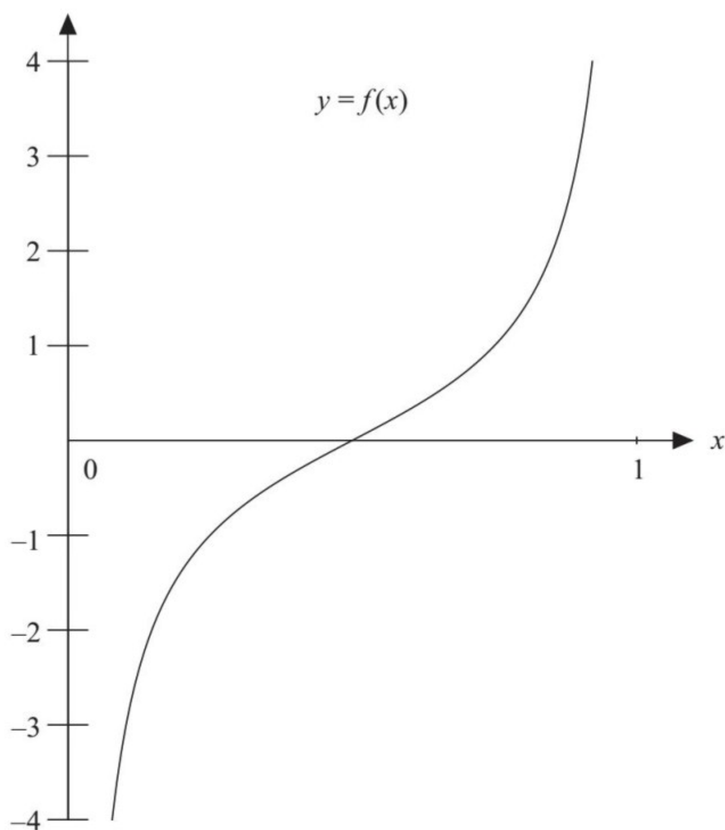


图8–1 双映射

对于了解三角函数的读者来说，函数$f:(0, 1) \to \mathbb{R}$可以通过$f(x) = \tan((x - 0.5)\pi)$来画图。

一旦我们知道在（0，1）和$\mathbb{R}$之间存在一个双映射，我们就知道它们拥有相同的基数。因为$|(0, 1)| = 2^{\aleph_0}$，我们证明了$|\mathbb{R}| = 2^{\aleph_0}$。现在，我们知道$|\mathbb{N}| = |\mathbb{Z}| = |\mathbb{Q}| < |\mathbb{R}|$。

实数包含有理数和无理数。实数和有理数的基数之间的区别是由无理数导致的。无理数的基数是$2^{\aleph_0}$，这是一个比有理数的基数大的无限数。因此，尽管实数包含有理数和无理数，它们实际上都是无理数。这是康托尔研究中的另一项惊人结论。

对于“实数的基数大于自然数的基数”，康托尔使用对角论证法给出了更加简单的证明。接下来，我们来研究一下这个论证法。

对角论证法

如果一个集合拥有基数$\aleph_0$，就可以被称为一个可数集合。这是因为如果$|S|=\aleph_0$，在集合S和自然数之间一定存在一个双映射。S的每一个元素都能够匹配一个唯一的自然数。我们用s_n表示与自然数n配对的S中的元素。我们可以这样描述集合S S， $=\{s_1, s_2, s_3, \ldots\}$。我们对$S$的元素进行了排序，所以我们得到了第一个元素，第二个元素，以此类推。

如果一个集合S拥有的基数大于$\aleph_0$，可以被称为不可数集合。我们无法用$\{s_1, s_2, s_3, \ldots\}$的形式来描述它，因为这只能描述$\aleph_0$个元素的集合，所以$S$中的无限多个元素就会被忽略。实际上，每个元素都会被忽略。相比更大的基数，较小的基数可以被忽略。

康托尔的对角论证法说明了在（0，1）之间的实数集合是不可数的。

证明：康托尔首先假设（0，1）之间的实数是可数的，并且推导出了一个矛盾。我们首先假设（0，1）之间的实数可数。因此，我们可以用$\{r_1, r_2, r_3, \ldots\}$ 来表示这个集合。每个实数r_i都可以表示为一个十进制数。我们将使用下面的标记来表示这些十进制数。

$$r_1 = 0.a_{11}\,a_{12}\,a_{13}\cdots,\ r_2 = 0.a_{21}\,a_{22}\,a_{23}\cdots,\ \cdots,\ r_i = 0.a_{i1}\,a_{i2}\,a_{i3}\cdots,\ \cdots$$

竖着列出这些表达式：

$$r_1 = 0.\ a_{11}\ a_{12}\ a_{13} \cdots\ a_{1i} \cdots$$

$$r_2 = 0.\ a_{21}\ a_{22}\ a_{23} \cdots a_{2i} \cdots$$

$$r_3 = 0.\ a_{31}\ a_{32}\ a_{33} \cdots a_{3i} \cdots$$

$$\cdots$$

$$r_i = 0.\ a_{i1}\ a_{i2}\ a_{i3} \ldots a_{ii} \ldots$$

$$\cdots$$

现在，构建一个实数 $b = 0.b_1\ b_2\ b_3 \cdots b_i \cdots$。如果 $a_{11} = 4$，令 $b_1 = 5$，否则令 $b_1 = 4$。如果 $a_{22} = 4$，令 $b_2 = 5$，否则令 $b_2 = 4$。一般来说，如果 $a_{ii} =4$，令 $b_i =5$，否则令 $b_i = 4$。我们研究列表中的对角元素，选择 b_i 来区分第 i 个对角数。

首先，请注意b的十进制展开只包含数字4和5。因此，b不是拥有两种十进制展开的实数。b是一个（0，1）之间的实数。因为b是（0，1）之间的实数，根据假设，它一定在该列表中。

但是，因为b_1不等于a_{11}，我们知道$b \neq r_1$。因为b_2不等于a_{22}，我们知道$b \neq r_2$。一般来说，因为b_i不等于a_{ii}，我们知道$b \neq r_i$。这个论证证明了b不在列表中。

我们已经证明b既在列表中又不在列表中，推导出了一个矛盾。所以，最初的假设一定是错误的，（0，1）之间包含的实数是不可数的。

连续统假设

康托尔证明了$\aleph_0$是小于$2^{\aleph_0}$的，即自然数的基数小于实数的基数。他考虑了一个明显的问题：在$\aleph_0$和$2^{\aleph_0}$之间是否存在基数？这等价于，是否存在某个实数的子集拥有大于$\aleph_0$，但小于$2^{\aleph_0}$的基数。他不认为存在这样的基数，但是无法证明这点。在$\aleph_0$和$2^{\aleph_0}$之间不存在基数的假设就是所谓的连续统假设。

1900年，希尔伯特列举了23个他认为新世纪最重要的问题。第一个问题就是连续统假设。

1940年，哥德尔证明了连续统假设无法被集合理论的定理推翻。1963年，保罗·科恩证明了无法通过定理证明连续统假设。这证明了连续统假设是独立于公理的——这是一个无法通过公理证明或反驳的陈述。

计算的基数

我们把注意力重新转到机器和计算上，看看计算机能够完成的所有可能计算的基数。

首先，我们将研究有限自动机能够完成的计算。一个计算包括一个有限自动机和一个有限输入。因为我们能够无限次地设计有限自动机，并且无限给予它们输入，有限自动机能够完成的计算一定也是无限的。这意味着，计算的基数至少是0。

我们介绍了给定任意有限自动机M和有限输入字符串I，如何编码$\langle M, I\rangle$中包含的信息片段。这个编码总是以4个1开头的、由0和1组成的字符串。我们研究的例子如下：

$\langle M, I\rangle = 11110011101110010110100111110010110$

毫无疑问，我们可以将这个字符串当作一个数字。如果我们在适当位置加入空格，$\langle M, I\rangle$就可以被视作一个相当大的数字

$$111\ 100\ 111\ 011\ 100\ 101\ 101\ 001\ 111\ 110\ 010\ 110$$

用这种方式来思考，编码给出了一个从有限自动机和有限输入到自然数的函数。有限自动机完成的计算包括自动机和有限输入。现在，我们拥有了从计算到自然数的函数。这意味着在计算集合和自然数子集之间存在一个双映射。因此，能够由有限自动机完成的计算集合的基数最大为$\aleph_0$。

我们证明了能够由有限自动机计算的基数最大为$\aleph_0$，最小为$\aleph_0$。我们得出结论：有限自动机能够完成的所有可能计算的集合拥有的基数为$\aleph_0$。

我们为有限自动机做出的全部论证都可以拓展到图灵机及其计算。给定一个图灵机和输入字符串，我们能够编码信息，并且将这个字符串视作自然数。我们从这一论证得到的结论是相同的。图灵机能够完成的所有可能计算的集合拥有的基数为$\aleph_0$。

任意计算机能够完成的计算同样也能由图灵机完成。因此，计算机只能完

成可数数量的计算。

可计算数

图灵在1936年发表的论文的第一行就是“可计算数或许可以像实数一样被简单描述，实数的十进制表达式可以用有限手段计算”。他还没有定义图灵机的概念，但是他所谓的“有限手段”就是使用图灵机或算法。

可计算数是实数，对这个实数而言，存在一个图灵机和输入，当图灵机在该输入上运行时，图灵机会在磁带上输出这个数字的十进制展开。图灵不只对有限的十进制展开感兴趣——这将会把可计算数限定到有理数。他希望图灵机的输出包括无理数。通过描述与我们介绍过的机器不同的图灵机，他实现了这个目的。图灵机没有接纳状态和拒绝状态，它们能够一直运行。为了输出一个无理数，他允许这个机器一直运行——只要它能够一直输出正确数字。

现在，我们使用本书中介绍的方法定义图灵机。我们的机器必须一直在接纳状态停止，以完成一次计算。当一次计算被接纳时，磁带上只会出现有限数量的符号。我们需要一点小技巧来处理无限十进制展开。

我们定义一个实数是可计算的，如果存在一个图灵机（或算法）以任意自然数n为输入，在有限次步骤后停止，c的十进制展开对应了n个十进制位置。在这个方法中，计算总会停止，但是输入数字n，我们也能得到尽可能准确的答案。

图灵证明了所有有理数都是可计算的。他证明了类似$\sqrt{2}$、π和e这样无理数都是可计算的。实际上，你很有可能永远都遇不到一个无法计算的数字。我们将证明并非所有实数都是可计算的，因此我们需要找到一个具体的例子。

一个非可计算数

在上一章中，我们证明了移位函数的最大值$S(n)$不是可计算函数。我们将使用这个结论来构建非可计算数。

我们将定义一个数字N，令$0 < N < 1$。N的十进制展开将包含完整的0和1。1的位置由$S(n)$给出。通常，我们定义 $N = \sum_{n=1}^{\infty} \frac{1}{10^{S(n)}}$

回忆一下，$S(1) = 1$，$S(2) = 6$，$S(3) = 21$。因此，

$$N = \frac{1}{10^1} + \frac{1}{10^6} + \frac{1}{10^{21}} + \cdots = 0.1000010000000000000001000 \cdots$$

下一个1出现在第107个小数位($S(4) = 107$)。

如果我们能够计算N，那么我们就能计算1在十进制展开中出现的位置，然后我们就能计算$S(n)$。但是，我们已经证明了$S(n)$不是可计算函数。因此，N不能是可计算数。

数字N得到了很好的定义，即便它不是可计算数。存在“你能够定义，却无法计算的数字”这一事实，是图灵发现的一个有趣结论。

非可计算数似乎相当稀少，但是这并非真相。我们将证明可计算数是可数的，也就是说，实际上所有实数都是非可计算数。

存在可数数量的可计算数

我们让$\mathbb{K}$表示可计算数。因为$\mathbb{K}$包含$\mathbb{Q}$，而且我们知道有理数的基数是$\aleph_0$，我们可以推导出$\aleph_0 \leqslant |\mathbb{K}|$。

我们还知道，图灵机能够完成的所有可能计算的集合是$\aleph_0$。这个可计算数的集合是所有可能计算的子集。因此，我们就有了$|\mathbb{K}| \leqslant \aleph_0$。

这两个不等式告诉我们，可计算数的集合是可以计算的，即$|\mathbb{K}| = \aleph_0$。

可计算数无法有效枚举

我们已经证明了可计算数是可数的。因此，在它们和自然数之间存在一个双映射。图灵证明了，尽管一定存在一个双映射，但无法使用图灵机去发现这个双映射。没有算法或有效的过程能够列出可计算数。这就是“可计算数无法有效枚举”的含义。

我们将只会考虑那些大于0并且小于1的可计算数，给出图灵对这个集合的证明。这个可计算数的子集是可计算的。这再次意味着它们和自然数之间存在一个双映射。我们可以使用这个双映射将这些可计算数列为$\{c_1, c_2, c_3, \ldots\}$，其中$c1$是与1匹配的可计算数，$c_2$是与2匹配的可计算数，以此类推。现在，我们进行康托尔的对角论证。首先，我们列出下面的十进制展开：

$$c_1 = 0.a_{11}\ a_{12}\ a_{13}\ldots a_{1i}\cdots$$

$$c_2 = 0.a_{21}\ a_{22}\ a_{23}\ldots a_{2i}\cdots$$

$$c_3 = 0.a_{31}\ a_{32}\ a_{33}\ldots a_{3i}\cdots$$

$$c_i = 0.a_{i1}\ a_{i2}\ a_{i3}\ldots a_{ii}\cdots$$

我们能够以完全相同的方式构建一个实数$b = 0.b_1b_2b_3\ldots b_\mathrm{i}\ldots$。如果$a_{11} = 4$，令$b_1 = 5$，否则令$b_1 = 4$。如果$a_{22} = 4$，令$b_2 =5$，否则$b_2 = 4$。一般来说，如果$a_\mathrm{ii} = 4$，令$b_\mathrm{i} = 5$，否则令$b_\mathrm{i} = 4$。我们观察表中的对角元素，选择$b$i来区分第$i$个对角数字。

数字b是介于0和1之间的实数，无法出现在列表中。这个列表包含了所有可计算数。这意味着b不是一个可计算数。但是，它为什么不是可计算数？我们可以用一个算法来描述从该列表构建b的过程：读入表中第一个元素，看第一个小数位的数字。如果它不是4，输出4。如果它是4，输出5。用第二个元素重复这一过程，再去看第二个小数位，以此类推。我们知道，b并不在可计算数的列表中。唯一的结论是可计算数的列表不是能够通过计算机构建的。如果列表能够通过计算机构建，那么b就是可以计算的，我们将得到一个矛盾。如果不能使用计算机构建列表，那么就不存在矛盾。因此，b一定不是可计算数。

图灵论文的题目是《论可计算数及其在判定问题上的应用》。图灵想要证明希尔伯特的判定问题是错误的。他首先需要给出有效过程或算法的定义。他通过图灵机进行了定义。之后，他需要找出无法被算法回答的问题。他定义了可计算数，并且证明它们是可计算的。再之后，他证明了没有算法能够证明它们是可计算的。最后，他证明如果存在这样的算法，我们就可以使用它来计算不在表中的对角数，这就证明了不存在算法能够列出这些数字。但是，这与该列表包含所有可能的可计算数相矛盾。

图灵证明了史上最伟大的数学家是错误的，这是一项重要成就。然而，真正的智慧蕴藏在论证方法中。他的理论机器简单易懂，他还证明了这一机

器能够进行任何计算。他的论证证明了通用计算机的存在，还证明了这个机器的能力和限制。

这篇论文奠定了理论计算的基础。

---

1. 如果A的每个元素都是B的元素，集合A被包含在集合B中。如果B包含A，那么B严格包含集合A，但是，至少有一个B中的元素不在A中。A中的每个元素都属于B，但是，有一些B中的元素不属于A。
2. Aleph，ℵ，是希伯来语字母表的第一个字母。
3. 这基于一个假设：如果$|A| G |B|$并且$|B| G |A|$，那么$|A| = |B|$。这一点是正确的，但对无限集合而言不够明显。这里产生的一个著名结论就是所谓的康托尔–伯恩斯坦–施罗德定理。
4. 读者可能在研究积分的限制时遇到这个想法。$\infty - \infty$和∞ / ∞被称作不确定形式。包括不确定形式在内的限制必须被仔细分析。

第九章 图灵的遗产

潜艇能游泳吗？

——艾兹格·迪科斯彻（Edsger W.Dijkstra）注

我们研究了图灵论文中包含的思想，指出这篇论文已经成为计算机科学的基石。但是，当图灵撰写这篇论文的时候，它不过是数学逻辑的结论，计算机科学还没有成为一门学科，现代计算机还没有被发明出来。在本章，我们将关注1936年论文发表到1954年图灵去世的18年间发生了什么。

首先，我们将简单地谈谈麦克斯·纽曼（Max Newman）。当我们谈论重要思想家的研究的时候，很容易低估其他人在他们的研究中扮演的角色。一位活跃的、能够给予学生帮助的导师会起到关键作用。纽曼就在自己的职业生涯中为图灵扮演了这样的角色。

图灵在剑桥大学国王学院的时候，纽曼是那里的讲师。1935年图灵学习的哥德尔非完备性定理课程正是由纽曼教授的。在这一课程中，图灵第一次学到了判定问题的概念。一年之后，纽曼意识到，图灵研究计算的方法拥有极高的原创性和重要性，并且鼓励图灵发表论文。纽曼致信伦敦数学协会的编辑，解释图灵做了哪些工作，并且敦促编辑发表图灵的论文。

第二次世界大战期间，图灵和纽曼都在布莱切利园研究密码破译。战后，纽曼在曼彻斯特大学创立了皇家计算机器实验室，并且邀请图灵加入这一团队。从1948年起，图灵开始在曼彻斯特工作，直到1954年离世。

纽曼同样帮助图灵下定决心前往普林斯顿进修，与阿隆佐·邱奇一同进行研究。

图灵在普林斯顿大学

新泽西州的普林斯顿同时拥有普林斯顿大学和普林斯顿高等研究院。1936年，普林斯顿大学数学系和高等研究院在同一栋建筑中办公。阿隆佐·邱奇当时是普林斯顿大学的教授，阿尔伯特·爱因斯坦和约翰·冯·诺依曼都曾是普林斯顿高等研究学院的成员。库尔特·哥德尔在1934年和1935年造访高等研究院，并于1939年成为终身成员。普林斯顿高等研究院是数学领域的中心，尤其是数学逻辑的中心。

1936年，邱奇33岁，是一位建树颇丰的逻辑学家。克莱尼和罗瑟都是邱奇的学生，都在数学逻辑领域发表了很多重要结论。当时的图灵只有24岁，刚刚开始自己的数学生涯。在剑桥大学，没有人能够与他共事。因此，他前往普林斯顿去拜访邱奇。

图灵并没有博士学位。当时，在英国参与学术研究不需要博士学位。但是，在咨询过纽曼和邱奇后，他认为进行博士学习是他前往普林斯顿的一个不错的理由。图灵在1936年9月来到普林斯顿。

在普林斯顿的时候，图灵结识了冯·诺依曼。1937年，冯·诺依曼写信支持图灵申请奖学金。在信中，他这样谈到图灵："他在我感兴趣的数学分支学科——概周期函数理论和连续群理论取得了不错的成绩。"注冯·诺依曼没有提到图灵对计算的研究，看来当时他还不知道图灵的论文。不久之后，他一定听说了这篇论文。1939年，斯塔尼斯拉夫·乌拉姆（Stanisław Ulam）回忆自己在1938年与冯·诺依曼讨论图灵的研究时写道："1939年，冯·诺依曼多次在与我讨论建立形式数学系统时提到图灵的名字。"注

1938年，图灵获得了博士学位，他在博士论文中介绍了顺序逻辑的研究。注作为研究的一部分，他发明了oracle，一个能够即时回答问题的黑盒。它本应是一个不可判定的问题。oracle机器是与oracle连接的图灵机。图灵只介绍了oracle的概念。这不是他论文的主要内容，而且他也从未详细研究过这一概念（波斯特跟进了这一概念，并在20世纪40年代证明了关于oracle的重要结论）。事实证明，这些机器同样是理论计算机科学的重要组成部分。正如罗伯特·索尔提到的，你可以将自己的笔记本电脑视作图灵机，将oracle视作网络。

在图灵完成博士学习后，冯·诺依曼邀请图灵担任他的助理，但是图灵婉言谢绝，并决定返回英国。

在图灵攻读博士学位期间，另一篇突破性论文被发表。这就是克劳德·香农（Claude Shannon）撰写的《逻辑和开关切换电路》。

克劳德·香农

1936年，克劳德·香农从密歇根大学毕业，获得电子工程学士和数学学士双学位。随后，他进入麻省理工学院攻读研究生。在麻省理工学院期间，香农研究了早期的模拟计算机。这项研究使他开始考虑开关和数字计算。

开关拥有"开"和"关"两种性质。很明显，它们能代表0和1。但是，香农证明了它们能做的事不止于此：所有布尔代数和二进制算术都能够使用电子

电路和开关实现。1937年，他在自己的硕士论文中提及这些结论，并在一年后发表了自己的论文。注

这篇论文成为计算机设计的基石。很快，这篇论文在电子工程师之间广为流传，影响了后来所有数字计算机的设计。

第二次世界大战

离开普林斯顿后，图灵返回剑桥工作，同时加入政府密码和暗号学校（Government Code and Cipher School，简称GC&CS）注，着手研究密码破译。1939年9月，英国与德国宣战后，图灵来到布莱切利园——GC&CS战时所在，全职进行解密工作。

德国当时使用恩尼格码机加密和解密所有军事信息。这种机器有一个键盘，操作员可以使用键盘输入字母。它的内部包含很多转子。在加密过程伊始，操作员必须以特定顺序插入转子，随后将它们旋转到特定的初始设置。操作员还需要连接一个插接板，作为初始设置的一部分。

一旦机器被配置好，操作员就开始输入消息。随着每个字母被输入，转子旋转，背光显示屏上就会显示出加密版的字母，操作员将这些字母记录下来。完成全部信息加密后，操作员使用标准摩斯电码，把加密信息发送给接收者。

编码拥有自反性质，接收者会按照同样的标准对机器进行初始设置，并且输入被加密的消息。整个过程结束后，他就可以得到加密前的文本。

破解这个密码分两步。第一步是精确理解恩尼格码机的内部构造。这些转子如何改变每个被输入的字母的编码？转子之间如何连线？插接板如何工作？在了解插接板的连线如何工作后，第二步就是找出每条被拦截消息的初始设置。

在战争开始前，波兰人完成了两个步骤的工作。波兰数学家马里安·雷耶夫斯基（Marian Rejewski）和他的密码学专家团队成功完成了恩尼格码机的逆向工程，并且研究出了完整的线路图。这是一项需要排列组合抽象性质知识的伟大成就。统计学和概率通常是密码学专家的工具，但是雷耶夫斯基意识到，一个名为“群论”的纯数学理论能够帮助他们判断该机器如何运转。他的团队提出的理论方法非常成功。

波兰密码学专家还发现了一个弱点，即初始设置会出现在消息的开始部分。这意味着有可能只通过加密消息的前6个字母判断初始转子设置。你

只需要遍历由所有可能的转子顺序、所有可能的转子设置和所有可能的插接板连线生成的所有6个字母的可能序列。问题在于这个可能数字太过庞大。战争开始时，这台机器拥有10^{22}种可能的初始设置。假设你每秒可以检查100万种设置，仍需要1亿年来检查完这些可能性。

雷耶夫斯基和他的团队使用两种方法来解决这个问题。他们意识到，他们需要使用群论概念来限制需要检查的可能性数量，而且他们需要一台能够检查这个受限列表的机器。他们设计了一台炸弹机来进行搜索。他们的机器的本质是将6台恩尼格码机连接起来，使它们一同运转，搜索可能设置的受限列表，直到发现匹配。

截至1938年，波兰能够解密拦截到的大部分军事信息。在1939年9月德国入侵波兰前，英国政府已经了解到波兰密码学专家取得的成绩。波兰人邀请英国密码学专家来到华沙，他们在那里解释了所有已知信息。

德国人逐渐将他们的加密技术变得更加安全。这些机器仍然使用三个转子，但是有5种可能的转子被安装在三个位置中的任意一个。初始设置的传输被更改，以消除波兰人发现的薄弱环节。在不同时间，不同的军事部门会分别进行调整。但是，海军是优先级最高的。德国密码学专家致力于确保海军恩尼格码机是最安全的。雷耶夫斯基的方法无法破解这些信息。

图灵在这时加入进来。他开始研究来自德国海军的加密信息。图灵利用了德国系统中的另一个弱点，即每个字母必须被加密为另一个不同的字母，没有字母能够被加密为自己本身。因此，如果你在信息中发现了自己非常确信的词汇，你就可以将这个词汇的字母在加密信息字母上滑过，直到你发现没有字母匹配。这个字母序列就是你的词汇的一种可能加密形式。这一理念以及群论和概率论理念，使他设计出了一台连接多个恩尼格码机复制品的机器，工作方式依旧是在一个受限列表中执行搜索。他的第一台炸弹机注在1940年春投入运行。

1942年，在美国参战后，图灵前往美国与那里的密码学家合作，协助设计了美国版的炸弹机。在美国期间，他造访了位于下曼哈顿地区的贝尔实验室，在那里他见到了克劳德·香农。除了研究密码破译，他们还讨论过对计算的研究。注

战争结束前，美国和英国有大量炸弹机运转。及时破解德方信息使盟军能够定位德军的舰艇航线，选择何时、在什么位置进行攻击。

图灵和雷耶夫斯基不仅在赢得战争的过程中做出了重要的贡献，还开启了密码学发展的新纪元。从此以后，密码学能够参考复杂数学，并且需要大

量的计算。

战争期间，麦克斯·纽曼也在布莱切利园，他领导破解了另一台德国加密机器——Tunny（金枪鱼）。Tunny采用的加密方法比恩尼格码机更加复杂。它被设计用于德军高层向前线将军发送信息。纽曼需要比炸弹机更强大的东西来破解这些密码。图灵将他介绍给汤米·弗劳尔斯（Tommy Flowers）。弗劳尔斯是使用真空管的专家。[注]当时，人们认为真空管太不稳定，无法在机器中大量使用。但是，弗劳尔斯证明了常识是错误的。他的机器被称作Colossus（巨人），这台机器可靠、速度快而且强大。这证明了计算的未来应该使用电子机器，而非电动机械机器。

战争末期，他们制造了11台这样的机器。其中两台被保留用于密码破解，另外9台机器连同它们的文档，因为可能的安全风险被政府销毁。[注]

1945年，因为在密码破译方面的贡献，图灵被乔治国王授予大英帝国勋章[注]。纽曼认为，图灵应该得到更多认可，6个月后，他同样被授予这一荣誉，但是他拒绝接受以示抗议。

尽管Colossus的设计成为秘密，计算机器能被制造的事情被更多人知晓。世界不同地方的许多团队开始设计早期的计算机。

在布莱切利园的工作结束后，纽曼和图灵开始重新研究新机器的设计。纽曼从一台被拆除的Colossus中保留了一些组件，来到曼彻斯特大学创立了计算机器实验室；图灵加入了伦敦的国家物理学实验室，设计自己的机器。

20世纪40年代的计算机发展

20世纪40年代，计算机从复杂可编程计算机器发展为真正的现代计算机。主要的进步表现为：设计机器从专门用于特定类型计算的机器变为通用计算机；开关从电动机械设备变为电子设备；存储程序式计算机中的程序和数据能够被读入内存。

关于“谁制造了第一台计算机”这个问题，答案取决于你对计算机的定义。大多数人赞同这一定义应该包括机器是通用的。如果你认为机器理论上不是通用的，但在实践中被设计为通用的，似乎也合情合理。例如，康拉德·楚泽（Konrad Zuse）在1941年制造了一台名为Z3的机器。它被设计用于特定计算。在原本的设计目的中，它不是通用机器，但在1998年，事实证明，以某种方式操纵输入，它实际上能够完成任何计算。楚泽的Z4被设计为通用计算机。

相比于试图将这一荣誉赋予某个人，介绍向着现代存储程序式电子计算机发展的主要成就似乎更具实际意义。

克兰德·楚泽

第二次世界大战成为制造计算机器的主要推手——亟须解决的问题不只包括密码，弹道表和核武器设计同样需要进行大量的计算，单凭人类计算者无法实现。轴心国和同盟国都开始设计机器进行计算。德国工程师康拉德·楚泽制造了一系列复杂、新颖的电动机械机器。楚泽并不了解香农或图灵的研究，在完全独立地进行研究。1945年，他制造的Z4是第一台被设计成通用机器的通用计算机。战争即将结束前，Z4在柏林投入使用。由于同盟国加强了对柏林的轰炸，这台机器被打包运往哥廷根。一段时间后，楚泽得以对其继续研究，1950年，Z4重新投入运行。

莫奇利和艾克特

朝着设计现代计算机前进的下一步，是从电动机械设备转变为电子设备——从中继设备转变为真空管。第一台通用电子计算机是ENIAC，包含17 468个真空管。宾夕法尼亚大学的约翰·莫奇利和普雷斯伯·艾克特制造了这台机器。他们于1943年开始这项工作，1946年时这台机器能够正常运转。然而，这台机器并非存储程序式计算机。对ENIAC进行编程需要插入各种接线以及设置开关。

冯·诺依曼

冯·诺依曼对设计计算机过程中包括的智力问题颇感兴趣，但是他也需要实际制造出一台计算机。他参与了制造第一颗氢弹的工作，这需要进行大量的计算。这些计算模拟震荡波的影响，首先是来自引爆可裂变材料的传统爆炸，之后是来自需要引爆聚变反应的裂变爆炸。之所以需要严谨的设计，是因为炸弹需要达到聚变阶段，而且不会提前将自身炸得四分五裂。很明显，研究人员无法测试采用不同设计的真正炸弹，决策层决定通过在机器上模拟链式反应来进行测试。

冯·诺依曼与ENIAC的制造者莫奇利和艾克特紧密合作。他们与其他几位学者一起研究新一代计算机的设计应如何进行改进。冯·诺依曼强调了存储程序的概念——程序应该与数据的存储方式一致。

《关于EDVAC的报告草案》就是对该团队观点的总结。同为这一团队成员的荷曼·哥斯廷（Herman Goldstine）整合了冯·诺依曼的笔记，撰写了这篇论文。这些笔记原本留有空白，是冯·诺依曼准备插入索引的地方，但是

在哥斯廷整理的版本里，空白和索引都被忽略了。哥斯廷还将这篇论文命名为“关于EDVAC的报告草案”，看起来就像是冯·诺依曼一个人的成绩。整理完成后，他油印了24份拷贝，并且开始传阅这篇论文。1945年，这篇论文广为传播，影响了后来的计算机设计。这种存储程序式电子计算机逐渐成为人们熟知的冯·诺依曼结构。

莫奇利和艾克特因为没有被署名为联名作者而备感失落。冯·诺依曼不仅“夺走”了许多他们认为属于自己的想法，还将这些想法公之于众。他们正在计划为自己的想法申请专利，并且进行授权。现在，这些想法已经广为流传，申请专利也就成了泡影。注图灵的名字同样没有出现在这篇论文中，尽管缺少索引，存储程序的概念实际上是图灵提出的。正如斯坦利·弗兰克尔（Stanley Frankel）所写：

> 在1943年或1944年，冯·诺依曼对图灵论文的重要意义有充分的了解……冯·诺依曼向我提到了这篇论文，在他的敦促下，我仔细研究了这篇论文……我坚信，这一基本概念是属于图灵的——从这个意义上来看，并非巴贝奇、洛芙莱斯期望的那样。注

为了对角论证，图灵需要一个事实：程序能够被编码为数字字符串，我们可以用两种方法来思考这些数字字符串——作为数字和作为程序，这种理解是计算机设计的重要理念之一。这使我们能够将程序视作数据。例如，它使我们能够设计高级语言，在这些语言中，程序由编译器编译，而编译器可以将程序视作数据；它使我们能够构建计算机网络。乔治·戴森（George Dyson）写道：“正如图灵构想、冯·诺依曼设计的那样，存储程序式计算机打破了代表事物的数字和完成任务的数字之间的差异。我们的宇宙永远不会是一样的。”注

冯·诺依曼的“草稿”被传阅后，掀起了一场制造第一台存储程序式计算机的竞赛。“baby”（婴儿）是第一台问世的此类机器，问世时间为1948年。纽曼是该实验室的负责人，图灵在这一年加入了该实验室。但是，baby和后来的曼彻斯特Mark 1都是由弗雷德里克·威廉斯（Frederic Williams）和汤姆·基尔伯恩（Tom Kilburn）制造的；图灵和纽曼都没有在设计过程中担任重要角色。baby被设计成一个用于测试随机访问存储设备的原型机，这里的随机访问存储设备随后成为全尺寸计算机Mark 1的一部分。

1949年，许多团队已经搭建并运行全尺寸通用存储程序式电子计算机。曼彻斯特的Mark 1和剑桥的EDSAC的启用时间彼此相距不足1个月。几个月后，CSIRAC于澳大利亚悉尼被制造出来并开始运行。颇具讽刺意味的

是，所有这些机器都基于冯·诺依曼的《关于EDVAC的报告草案》，但是直到1951年，EDVAC才开始运行。

图灵测试

战后不久，图灵进入位于伦敦的国家物理实验室工作。在此期间，他决定设计自己的计算机。他将之命名为ACE（Automatic Computing Engine，自动计算引擎），并且做出了完整的设计。一个小团队开始着手制造ACE，但是工作进展缓慢。一台原型机最终于1950年制造成功并投入运行，但是同一时间已经出现了其他更加强大的计算机，甚至连图灵都加入了纽曼在曼彻斯特大学的团队，一直在那里工作到离世。

在曼彻斯特期间，图灵发表了一篇题为“计算机器和智能”的论文。这篇论文以“机器能思考吗”这个问题开篇。图灵思考了我们如何区分人或机器是否正在思考。他提出了“模仿游戏”，后来变成了所谓的“图灵测试”。在这个游戏中，一位人类仲裁身处与人类和计算机不同的房间。仲裁知道存在一台计算机A和一个人B，但是并不知道哪个是人，哪个是计算机。仲裁会向A和B提出问题，A和B进行回答。人类仲裁的目标是判断哪个是人，哪个是计算机。(注)

这是一篇非常有趣的论文。图灵的论证十分充足，并且考虑了各种对“机器能够思考”这个想法的反驳。这篇论文的原创性非常强。他的担忧之一是人类可能会通过心灵感应交流，他非常严肃地思考了这个问题，建议将所有人放在“防心灵感应”的房间里。

图灵的中心论点是，我们通过与人类交互，判断某个人是否认真，是否在思考。我们不会试图理解他们大脑里的神经元如何工作，但是会考虑我们是否有过一次有意义的对话。同样的道理也适用于机器。如果我们想要知道一台机器是否拥有智能，或是否拥有意识，我们应该通过交互进行判断。

值得一提的是，有一个版本的图灵测试已经成为我们日常生活的一部分。只有在这个版本中，计算机才试图区分人类和机器。CAPTCHA（Completely Automated Public Turing Test to Tell Computers and Humans Apart，全自动区分计算机和人类的图灵测试）经常会出现在在线表格中。在你提交表格前，你必须回答一个CAPTCHA测试题——一般包含读取一些畸形的文本，将字母和符号键入输入框。

机器思考的想法自然将我们引向了机器能否理解、能够拥有意识的想法中。支持和反驳两方面都有很多人。反对机器能够思考和理解的最著名的

论点就是中文屋论证。这是哲学家约翰·塞尔（John Searle）在1980年发明的思想实验，这个实验的根据是图灵测试。塞尔想象自己被放置在一个屋子里。他并不理解中文，但是他有一本告诉他在各种情况下应该做什么的书。人们可以通过一个小槽向屋里塞入纸张。塞尔可以在纸上写字，并且通过小槽向屋外人送出这些纸。屋外的人都是汉语母语者。他们在这些纸上用中文写下问题。塞尔观察这些消息。这对他而言是没有意义的，但是他能在自己的书里查阅这些符号字符串。当他找到对应的字符串时，书上写着如何在纸上写下答案，并把纸送出屋子。

外面的人是仲裁者。他们试图判断屋子里的人是否理解中文。因为他们一直能得到所提问题的正确答案，他们认为屋里有一个理解中文的人。但是，中文屋理解中文吗？或者，它是否只是假装理解中文呢？

这一论点发布之初就引发了大量关于“中文屋里是否发生了真正的理解”的争论。很明显，塞尔充当了一台通用计算机，书就是程序。我们看到，通用计算机非常简单，复杂的是编写程序。塞尔和书的“思考”就如同计算机和程序的“思考”。

思考包括什么？它是否意味着“拥有意识”？这些都是很难回答的问题。哲学家不知道对人类而言，拥有意识到底意味着什么。一些人甚至坚信，我们感受到的自由意愿是虚构的，我们感觉到的意识，不过是物理大脑的一个附带现象。

试图显式定义理解和模拟理解之间的差异，似乎超越了人类的能力范畴。

麻省理工学院计算机科学家斯科特·阿伦森（Scott Aaronson）注对此进行了精妙的总结：

> 人们能够找出很多重要的、有信服力的论证，去反驳“机器会思考”的可能性。这些论证唯一的问题在于，它们也在质疑“大脑会思考”的可能性。
>
> 一个流行观点是，如果一台计算机表现出智能，它只反映了对其编程的人类的智能。但是，如果人类的智能只是数十亿年进化过程的反应，又该如何解释？

陨落

图灵是一位同性恋者，在当时的英国，男同性恋是非法的。1952年，图灵的公寓被闯入，他向警察报告了这一事件。在调查过程中，警方确信，图灵与一位名为阿诺德·默里（Arnold Murray）的男性有过性接触，而阿诺

德正与此案相关。图灵承认与默里有过三次性交。这个数字足以使图灵获得6项罪名的指控，警方决定逮捕图灵和默里。

图灵非常担心成为囚徒。在审判中，他的辩护人提出，相比于被关入监狱，图灵应该接受同性恋治疗，并被判缓刑。法官同意了这一请求，判图灵缓刑一年，要求他接受器官疗法（器官疗法通常被称作化学阉割，包括接受合成雌性激素注射）。

被定罪后，图灵失去了自己的安全审查资格，无法继续为GC&CS就有关密码破译的问题进行咨询，但是他仍然保有自己在曼彻斯特的计算机器实验室的工作。两年后的1954年6月，图灵被发现死于氰化物中毒。

一个广为流传的说法是，图灵吃下了浸泡过氰化物的苹果。哲学家杰克·科普兰（B. Jack Copeland）在自己撰写的《图灵：信息时代的先驱》（*Turing: Pioneer of the Information Age*）一书中，仔细研究了围绕在图灵之死周围的诸多事件，得出结论：这是一次意外事故。

图灵是被他的保洁员发现的。遗体旁有被吃掉一半的苹果。这个问题苹果从未被检测过。调查图灵之死的警察确实发现了电解设备——一个装有连接电极液体的盘子。当警方发现这个盘子时，液体正在冒气泡。

图灵喜欢做一些化学实验，其中一些实验就用到了氰化物。他在卧室旁边有一个小房间，作为自己的实验室。他曾用电解法成功镀金了一把勺子。考虑到这些情况，事实更有可能是图灵在进行某个包含氰化物的危险实验时意外身亡。

科普兰同样研究了图灵死前几天的心理状态。他提到，激素治疗已经在一年前停止。他的工作进展顺利。在图灵生前最后一周与他有过交流的人认为，图灵精神状态不错，并没有消沉，他并未表现出任何要自杀的征兆。

对图灵遗体的检查草草结束。验尸官提到："我被强迫做出这是蓄意行为的结论。像他这样的人，从来没有人知道他接下来将会做什么。"注这似乎做出了自杀的假设，并没有真正考虑过其他可能性。

科普兰在书的结尾这样写道：

> 图灵之死的准确情况或许会永远变成不解之谜。不应该说图灵是自杀的——因为我们只是不知道真相。或许我们只会耸耸肩，同意没有最终的结论，并且关注图灵的一生和他惊人的研究成果。

道歉和赦免

尽管我们不能确定图灵之死的真相，但是很明显，他受到了英国司法系统惨无人道的粗暴对待。2009年9月，第二次世界大战爆发70年后，英国首相戈登·布朗（Gordon Brown）向图灵的遭遇正式做出道歉：

> 这是我们应该深思的一年——对英国而言，这是她作为一个国家纪念那些我们曾经深深亏欠的人们的机会。一系列纪念日和活动激起了我们的自豪感和感激心——正是这些塑造了英国人。今年早些时候，我同萨科齐和奥巴马两位总统一起，纪念65年前在诺曼底海岸奋勇作战的英雄们。就在上周，我们一起纪念了“英国政府拿起武器反抗法西斯，并宣布第二次世界大战爆发”70周年。因此，今天我怀着愉快与自豪的心情站在这里，由于一些计算机科学家、历史学家和LGBT（女同性恋、男同性恋、双性恋和跨性别者），我们今年才有机会纪念和庆祝在英国反抗独裁统治战斗中密码破译者艾伦·图灵做出的伟大贡献。
>
> 图灵是一位非常聪明的数学家，因为对德国恩尼格码密码的研究而闻名于世。毫不夸张地说，没有他的杰出贡献，第二次世界大战的历史或许会截然不同。他是在扭转战局过程中做出独特贡献的一个人。因此，他应得到的感激使他受到的非人道待遇更加令人扼腕叹息。
>
> 1952年，他被判严重猥亵——实际上，他被视作男同性恋。在被关入狱和接受治疗之间，他选择接受化学阉割——注射一系列雌性激素。仅仅两年之后，他的生命就永远地结束了。
>
> 成千上万人聚集起来，为艾伦·图灵寻求正义，要求政府承认图灵所遭受的非人道待遇。图灵接受当时法律宣判的时候，我们无法将时钟逆转回去，他受到的待遇当然是完全不公平的，我很高兴有这个机会，对他的遭遇表示我本人和我们的深深的歉意。正如图灵受到的指控一样，他和其他成千上万违反《反同性恋法》的男同性恋都受到了严酷的对待。许多年间，数百万人仍生活在对指控的恐惧中。我很骄傲地宣布，那些日子已经过去，过去12年来，政府做出了很多努力，让LGBT的生活变得更加公平、更加平等。认识到图灵作为英国最著名的“反同性恋”受害者的身份，是我们朝着公平迈出的重要一步。这一步，来得太迟。
>
> 但是，图灵对人类的贡献应该得到承认。对于我们这些1945年以后，出生在一个统一的、民主的、和平的欧洲的人来说，很难想象我们的大陆曾经经历了人类最黑暗的时期。很难相信人们能够变得对仇恨如

此着迷——被反犹太主义、反同性恋、仇外情绪和其他残忍的偏见冲昏了头脑，以至于毒气室和火葬场像过去数百年间曾经代表了欧洲文明的画廊、大学和音乐厅一样，成为欧洲大陆的一部分。

感谢那些像艾伦·图灵一样，义无反顾地投身反法西斯战争的男人和女人们，对大屠杀和全面战争的恐惧，已经成为欧洲历史的一部分，但并非欧洲的现在。因此，我代表英国政府，以及所有因图灵的贡献而自由生活的人们，自豪地说："（图灵）我们很抱歉，你应得到更好的对待。"

2013年12月24日，图灵得到了英国皇室的赦免。

---

1. 实际上，这段话来自迪杰斯特拉（Dijkstra）在1984年一次会议上的讲话："这一领域的开辟者一直十分令人困扰：约翰·冯·诺依曼认为计算机和人类大脑十分相似，这一想法堪比中世纪思想家；艾伦·图灵思考标准来设置判断机器能否思考的问题，这是我们现在知道的一个问题，与判断潜艇是否游泳相似。"

2. 安德鲁·霍奇斯。Op. cit.第167页。

3. 马丁·戴维斯。《通用计算机：从莱布尼茨到图灵》,第169页。

4. 因为图灵是邱奇的学生，他采用了λ积分来表示这个理念。不幸的是，这导致研究变得比较难懂，或许这解释了为什么它没有被广泛传阅。

5. 《对继电器和开关电路中的符号分析》，1938年。

6. 这个地方后来被更名为政府通信总部（Government Communications Headquarters），但是它更为人所知的缩写是GCHQ。相当于美国的国家安全局。

7. 英文单词"bombe"由波兰语"bomba"发展而来。至于波兰人为何称他们的机器为bomba（意为英文的"炸弹"）似乎成为一个谜团。当被问及这点时，据说雷耶夫斯基的回应是他想不出更好的名字。

8. 香农和图灵都对"使用来自概率中的理念"提取数据中的信息感兴趣。后来，香农延伸了一些战时的研究成果写出了一，篇突破性的论文《通信的数学原理》（*A Mathematical Theory of Communication*），是信息原理的基础之一。图灵撰写了多篇关于概率在密码学方面应用的文章。这些文章都曾是机密信息，现在才得以公布于众（2012年，两篇论文解密，可以在http://www.nationalarchives.gov.uk上找到）。

9. 英国使用“valve”（电子管）这个词，而美国使用“vacuum tube”。

10. 布莱切利园的国家计算博物馆制造了一台Colossus。访客能够看到这台机器运行。

11. OBE代表大英帝国勋章（Order of the British Empire）。这比爵士头衔低好几个级别。

12. 莫克利和埃克特多次上诉至法院，试图申请专利，但从未成功。

13. 安德鲁·霍奇斯，Op. cit.第382页。

14. 乔治·戴森，《图灵的大教堂》文前第9页。

15. 1990年，休·罗布纳（Hugh Loebner）承诺为第一台通过图灵测试的计算机提供100 chapter09美元奖金和金牌。自1991年起，每年都会举行一次比赛，“最接近人类的”电脑将被授予铜牌和一定奖金。

16. 斯科特·阿伦森，《自德谟克利特以来的量子计算》第45页。

17. 1954年6月18日，《每日电讯报》报道了此事。霍奇斯和科普兰的传记都对此进行了引用。

拓展阅读

图灵的论文

本书关注了图灵的《论可计算数》。这篇论文相当晦涩，因为它的目标受众是学术界人士。如果想要进行深度研究，我推荐各位读者阅读查尔斯·佩措尔德（Charles Petzold）的《带注解的图灵：艾伦·图灵关于可计算性和图灵机的历史性论文的导览》（*The Annotated Turing: A Guided Tour Through Alan Turing's Historic Paper on Computability and the Turing Machine*）。佩措尔德为图灵的论文增加了许多杰出的长篇注解，帮助读者准确理解图灵在每个阶段究竟研究了什么。佩措尔德同样介绍了这篇论文的相关历史。

图灵在《计算机器和智能》中讨论了模仿游戏（图灵测试）。这篇文章是为一般读者撰写的，在网上很容易找到。这篇文章非常值得一读。

对于那些想要深入研究图灵文章的读者来说，安德鲁·霍奇斯在http://www.turing.co.uk维护了大量信息。另一个在线资源是图灵档案馆，由杰克·科普兰和戴安·普劳德富特维护，网址：http://www. alanturing.net。

图灵的传记

许多人都为图灵写过传记，其中包括他的母亲和哥哥。但是，有两部传记比较突出：安德鲁·霍奇斯的《艾伦·图灵：恩尼格码》和杰克·科普兰的《图灵：信息时代的先驱》。霍奇斯的书更为全面，这两本书都是由真正理解图灵观点的学术作者完成的，能够很清晰地解释图灵的观点。

计算原理的历史

马丁·戴维斯的《通用计算机：从莱布尼茨到图灵》包含了奠定图灵机基础的数学和逻辑的历史与解释。

乔治·戴森的《图灵的大教堂》从戴维斯结束的地方起步，讲述了第二次世界大战后，计算机如何诞生的故事。这本书更关注冯·诺依曼，而非图灵。

沃尔特·艾萨克森（Walter Isaacson）的《创新者：一群黑客、天才和极客如何缔造数字革命》，从巴贝奇开始到互联网结束，全面介绍了计算机的发展史。

计算机，思维和宇宙

斯科特·阿伦森、大卫·多伊奇和道格拉斯·霍夫斯塔特（Douglas Hofstadter）是三位计算机科学家，他们进行了许多突破性研究，其中很多理念都与计算理论相关。阿伦森的《自德谟克利特以来的量子计算》，多伊奇的《无限的开端：转变世界的解释》，霍夫斯塔特的《哥德尔、埃舍尔、巴赫》都十分精彩。

元胞自动机

我们只是简单研究了元胞自动机，但是它们的历史更长，也更有趣。当最早的计算机被制造出来时，乌拉姆和冯·诺依曼最先对元胞自动机进行了研究。20世纪50年代，普林斯顿大学的尼尔斯·巴里切利（Nils Barricelli）使用计算机来模拟细胞交互。乔治·戴森的《图灵大教堂》对这一研究进行了精辟的解读。

1970年，约翰·康韦（John Conway）借助二维元胞自动机定义了生命。马丁·加德纳（Martin Gardner）在《科学美国人》杂志中对此进行了普及。威廉·庞德斯通（William Poundstone）的《递归宇宙》是一本关于这些自动机的历史和简单规则如何孕育复杂性的书籍。

史蒂夫·沃尔弗拉姆的《新科学》是一部关于一维元胞自动机的、带有大量注释的百科全书。

计算理论

杜德尼（A.K. Dewdney）的《新图灵选集》包含66个短章。每一章都是一篇与理论计算机科学重要部分相关的短文。它涉及了大量话题，包含了大量信息。对于那些想要深入研究理论计算机科学的人而言，这本书绝对值得一读。

现在，有很多关于计算理论的优秀文章，都是写给计算机科学本科生和数学专业学生的。迈克尔·西普赛的《计算理论导引》是一本不错的现代教科书。一本年头略久但更加精彩的书是约翰·霍普克罗夫特（John Hopcroft）和杰弗里·乌尔曼（Jeffrey Ullman）的《自动机理论、语言和计算导论》。

此外还有大量在线资源。ADUni.org就是其中比较出色的。